Fachhochschule Fulda
Fachbereich Angewandte Informatik

Projektdokumentation:

Simulation der Beschleunigung von Fahrzeugen

Prüfungsleistung Systemprogrammierung

WS 2004/2005

Prüfungskommission:

Prof. Dr. W. Heinzel
Prof. Dr. Siegmar Groß
Dipl.-Ing. Peter Klingebiel

Projektteilnehmer:

Thorsten Kohlhepp, Matr. Nr.: XXXXXX
to@arcor.de

Dominik Haug, Matr. Nr.: XXXXXX
dominik.haug@gmx.net

Inhaltsverzeichnis:

1 Einleitung:.....5

1.1 Projektdefinition.....5

1.2 Aufgabeverteilung der Projektmitglieder.....6

1.3 Projektumsetzung.....6

2 Projektdokumentation Teil 1.....7

2.1 Implementierung des Berechungsservers.....7

2.1.1 Das Berechnungsmodell.....7

2.1.2 Implementierung der parallelen Berechnung.....8

2.1.3 Die Kommunikation mit dem Simulationsprogramm.....9

2.2 Implementierung der Realtime-Schaubilder.....12

2.2.1 Visualisierung der Ergebniskommunikation.....13

2.2.2 Realtimeplotting der Simulationsergebnisse.....13

2.3 Implementierung der 3D Fahrzeug-Armaturen.....14

2.3.1 Die Zusammensetzung der 3D-Armaturen.....15

2.3.2 Die Tachometer (Geschwindigkeit und Drehzahl).....17

2.3.3 Die Displays (Stoppuhr und Gangzähler).....19

2.3.4 Die Animation der Armaturen.....21

3 Projektdokumentation Teil 2.....25

3.1 Grundlegendes.....25

3.2 Programmablauf.....26

3.2.1 Ablauf des Hauptprogramms.....26

3.2.2 Ablauf der Animation.....28

3.3 Datenkapselung.....29

3.3.1 Datenkapselung Fahrzeugdaten, Ergebnissdaten.....29

3.3.2 Datenkapselung Window3D Klasse.....30

3.4 Kontrollverbindung zu Berechnungsserver.....31

4 Benutzerdokumentation.....33

4.1 Installation und Inbetriebnahme.....33

4.1.1 Systemanforderungen.....33
4.1.2 Kompilieren der Programme.....33
4.1.3 Ausführen der Programme.....34
4.2 Verbindungsaufbau zu einem Berechnungsserver.....35
4.2.1 Kontrollfunktionen, Fernsteuerung des Berechnungsservers.....35
4.3 Laden eines Fahrzeuges.....36
4.3.1 Hinzufügen eines Fahrzeuges.....36
4.3.2 Aufbau der .car Dateien.....37
4.3.3 Ändern von Fahrzeugdaten.....38
4.4 Berechungen starten.....38
4.4.1 Gemischter Modus.....39
4.4.2 Graphen Modus.....39
4.4.3 Animationsmodus.....39
4.5 Realtime-Animationen.....40
4.5.1 Graphen Modus.....40
4.5.2 Animationsmodus.....40
4.5.3 Gemischter Modus.....41
4.6 Darstellungseigenschaften ändern.....41
4.6.1 Eigenschaften der Schaubilder ändern.....41
4.6.2 Eigenschaften der 3D Animation ändern.....42
4.7 Die Bedienung des Berechnungs-Servers.....43
4.7.1 Änderung der Sleeptime.....43
4.7.2 Beenden des Berechungsprogramms.....44

5 Anhang:.....45
5.1 Abbildungsverzeichnis:.....45
5.2 Mathematische Modellbildung:.....46
5.3 Klassendiagramm Berechungsserver.....48
5.4 Klassendiagramm Simulationsprogramm.....49

Vorwort:

Dieses Projekt wurde im Rahmen der Lehrveranstaltung Systemprogrammierung an der Fachhochschule Fulda entwickelt. Alle Sourcen sind dementsprechend unter der GNU Public License weiter verwendbar. Sollten bei der Umsetzung dieses Projektes Codefragmente verwendet werden, die nicht von den Entwicklern in eigener Arbeit erstellt wurden, so ist dies gekennzeichnet und muss auch bei weiterer Verwendung gekennzeichnet bleiben.

Jegliche Zitate aus externen Quellen, die in dieser Dokumentation genannt werden, sind als Zitate gekennzeichnet und deren Quellenhinweise sind im Anhang angefügt.

Bei dem hier beschriebenen Programm wird dringend empfohlen, bevor es angewendet wird, die Benutzerdokumentation zu lesen, zu finden in Kapitel 4.

Das Programm wurde ausschließlich in Java entwickelt und so konzipiert, daß es unter den Betriebssystemen Unix, Linux und Windows lauffähig ist, sofern die Java 3D Komponenten installiert sind. Weitere Informationen zum Starten und Verwenden des Programms finden sie in der Benutzerdokumentation in Kapitel 4.

Wir übernehmen keine Haftung für Schäden, die durch die Benutzung des Programms oder dessen Quellen entstehen. Die Benutzung und Weiterentwicklung des Programms geschieht ausdrücklich auf eigene Gefahr des Benutzers.

1 Einleitung:

Diese Projektdokumentation ist in drei Punkte gegliedert. In ersten Punkt werden allgemeine Dinge, wie Projektdefinition und die Aufgabenverteilung beschrieben. Die Punkte zwei und drei beinhalten die Detaildokumentationen der einzelnen Projekteilnehmer, in der die implementierten Programmteile von dem jeweiligen Entwickler dokumentiert werden.

1.1 Projektdefinition

Ziel des Projektes „Simulation der Beschleunigung von Fahrzeugen“ ist es, eine verteilte Software zu implementieren, die das Beschleunigungsverhalten von Fahrzeugen berechnet und simuliert. Dabei sollen die Berechnung und die Darstellung/Simulation der Ergebniswerte auf verschiedenen Rechner, die über ein Netzwerk miteinander kommunizieren, erfolgen. Das Simulationsprogramm soll, mit einer grafischen Benutzeroberfläche ausgestattet, dem Benutzer die Möglichkeit bieten, Berechungen an im Netz verteilte Berechungsprogramme (im folgenden als Berechnungsserver bezeichnet) zu senden. Über die grafische Benutzeroberfläche soll der Benutzer die Möglichkeit haben, Kommandos an die Berechungsserver zu senden, um die Berechungsserver fernzusteuern. Die Berechungen mehrerer Fahrzeuge soll parallel erfolgen. Während der Berechnung, sollen in bestimmten Intervallen, parallel zur Berechung, Zwischenergebnisse an das Simulationsprogramm zurückgesendet und dort dargestellt werden. Das Eintreffen dieser Zwischenergebnisse soll in Schaubildern erfolgen. Nachdem die Berechungen abgeschlossen und vollständig an das Simulationsprogramm gesendet wurden, soll der Benutzer die Möglichkeit haben, sich die Ergebniswerte in einer Realtimsimulation darstellen zu lassen. Die Visualisierung erfolgt dabei in Schaubildern, in denen sämtliche Ergebniswerte in Echtzeit dargestellt werden. Zusätzlich soll die Simulation durch eine 3D-Animation dargestellt werden können. Die Ergebniswerte sollen dabei durch ein Fahrzeugarmaturenbrett, mit sämtlichen Instrumenten zur Ergebnisdarstellung in Echtzeit visualisiert werden. Zusätzlich soll mit einem Blick durch die Windschutzscheibe und der bei der Fahrt vorbei ziehenden Landschaft, der Eindruck der Geschwindigkeit des Fahrzeuges vermittelt werden.

1.2 Aufgabeverteilung der Projektmitglieder

Die Aufgaben zur Projektumsetzung wurden wie folgt auf die einzelnen Projektmitglieder verteilt:

Teil 1, von Dominik Haug:

Implementierung der Berechnungsalgorithmen des Berechnungsprogramms.

Programmierung paralleler Berechnung mehrere Fahrzeuge gleichzeitig in einem Berechnungsserver (Threads).

Socket-Kommunikation mit Hauptprogramm und Berechnung der Simulation parallel. Implementierung der messagebasierten Kontroll-Kommunikation auf Berechnungsseite und Ergebniskommunikation.

Grafische Darstellung der Ergebniswerte im Hauptprogramm. Programmierung der Tachometer, Drehzahlmesser und Stoppuhren mittels Java3D sowie die exakte Ausgabe sämtlicher Simulationsergebnisse in Diagrammen mit Unterstützung von Realtimplotting (realisiert mit Timern).

Programmierung der Fahrzeugdateiverwaltung

Teil 2, von Thorsten Kohlhepp:

Programmierung der Benutzeroberfläche

Kommunikation zwischen Steuerungs- und Berechnungsprogramm aus Sicht der Steuerung. Parallele Verarbeitung der Ergebniswerte der Berechnung für Visualisierung und Empfang weiterer Ergebniswerte.

Animation der Fahrzeugbeschleunigung (Modellierung der Außenwelt mittels Java3D mit Sicht durch die Windschutzscheibe.

„Zusammenbau“ der Java3D Szene (einbinden der Armaturen)

1.3 Projektumsetzung

Die Implementierung des Projektes basiert vollständig auf der Programmiersprache Java, von Sun Microsystems. Die Visualisierung der 3D Komponenten der Fahrtsimulation aus Sicht des Fahrers, wird mit der Programmierschnittstelle Java 3D realisiert.

Für die Kommunikation der verteilten Darstellungs- und Berechungsprogramme untereinander, wird als Netzwerk-Transportprotokoll das TCP/IP eingesetzt, was mit Hilfe von Java-Sockets realisiert wird.

2 Projektdokumentation Teil 1

Im folgenden Dokumentationsteil werden Programmteile dokumentiert, die von Dominik Haug erstellt wurden. Dazu gehören die Implementierung des Berechnungsservers, mit sämtlichen Routinen zur Kommunikation mit dem Simulationsprogramm, die Implementierung der Realtime-Schaubilder zur E rgebnisdarstellung und die Programmierung der 3D Fahrzeugarmaturen und deren Animationen.

2.1 Implementierung des Berechungsservers

Der Berechungsserver dient dem Simulationsprogramm als Berechungsunterstützung, an den die Berechungen ausgelagert werden können. Durch diese Auslagerung steht dem Simulationsprogramm eine höhere Performance zur Verfügung. Durch die Verteilung der Berechungen auf andere Computer, können größere Mengen an Simulationen berechnet werden. Da die Berechnungsserver lediglich den Dienst der Berechung anbieten, wurde hier auf eine grafische Oberfläche verzichtet. Über eine Konsole können jedoch Benutzerbefehle entgegengenommen werden (siehe Benutzerhandbuch). Im folgenden Punkt soll kurz auf das Berechungsmodell eingegangen werden.

2.1.1 Das Berechnungsmodell

Die Startbedingung einer Beschleunigungsberechung, ist ein Fahrzeug mit eingelegtem ersten Gang und der Fahrt bei niedrigster Drehzahl. Auf diese Art kann auf die sehr komplizierte Simulation des Einkuppelns des ersten Ganges (schleifende Kupplung, Abhängigkeiten des Verhalten des Fahrers) verzichtet werden.
Die Berechung erfolgt durch eine Differenzialgleichung zweiter Ordnung. Auf die exakte mathematische Modellbildung mit sämtlichen Parametern, soll in dieser Dokumentation nicht eingegangen werden, diese ist im Anhang dieser Dokumentation zu finden (Punkt 5.1). Die Differentialgleichung zweiter Ordnung wird dabei durch ein Integrationsverfahren berechnet. In jeder Iteration wird dabei mit der aktuellen Fahrzeuggeschwindigkeit über das Getriebe die aktuelle Drehzahl bestimmt. Mit dieser Drehzahl wird das Drehmoment des Motors berechnet und wiederum über das Getriebe die Antriebskraft bestimmt. Diese Antriebskraft ergibt zusammen mit der Rollreibungskraft und der aus der aktuellen Geschwindigkeit resultierenden Luftreibungskraft die Gesamtkraft die auf das Fahrzeug wirkt. Mit dieser Gesamtkraft

lässt sich dann die neue Geschwindigkeit berechnen. Der Kreislauf dieser Iteration ist in Abbildung 1 dargestellt.

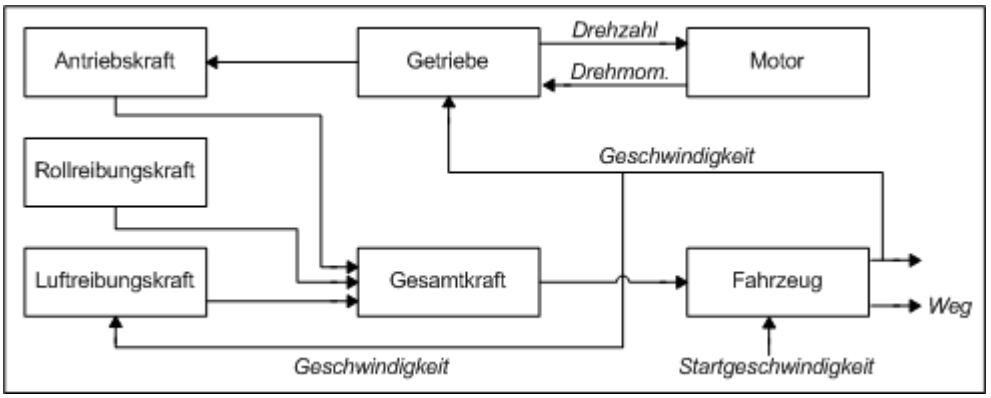


Abbildung 1: Berechnungsmodell, Iterationskreislauf

Je kleiner dabei der Iterationsabstand ist, desto genauere Simulationen lassen sich berechnen. Der Parameter der Iteration kann in der Klasse *GlobalData* des Berechungsserver eingestellt werden.

2.1.2 Implementierung der parallelen Berechnung

Ein Berechungsserverprogramm kann mehrere Fahrzeugsimulationen parallel berechnen. Die Berechung eines Fahrzeuges erfolgt dabei in einem eigenen Thread:

```
public class Calculation extends Thread
```

Sendet ein Simulationsprogramm mehrere Fahrzeuge an den Berechungsserver wird für jedes Fahrzeug ein eigener Berechungssthread gestartet:

```
for (Enumeration i = carDataVector.elements (); i.hasMoreElements ();)
{
  CarData carData = (CarData) i.nextElement ();
  calculation[k] = new Calculation (carData, host);
  calculation[k].start ();
  k++;
}
```

Die Fahrzeugdaten werden dabei als serialisierter Vektor (`carDataVector`) aus `CarData` Objekten vom Simulationsprogramm empfangen. In der `run()`-Methode der Berechnungs-Threads wird die Iterationsberechung solange in einer Schleife durchlaufen, bis der Endzustand einer Simulation erreicht ist (maximale Drehzahl, im

höchsten Gang) oder die maximale Anzahl an Iterationen erreicht wurden (falls Fahrzeug aufgrund der Motor/Getriebe-Eigenschaften den Endzustand nie erreicht):

```
do
{
  //Berechne eine Iteration
}
while (((car.getGearing ().gibGang () < car.getGearing ()
      .gibGangZahl ()) || (car.getGearing ().gibDrehzahl () < car.getGearing
      ().gibSchaltDrehzahl ()))  &&  (k < GlobalData. maxIter));
```

Nach jeder Iteration werden sämtliche Simulationsdaten in einem Ergebnisobjekt (Klasse `SimResult`) gespeichert. Diese Ergebnisobjekte werden nach jeder Iteration einem Ergebnisvektor hinzugefügt (Klasse `SerVector`). Hat dieser Ergebnisvektor eine bestimmte Größe erreicht (definiert in Klasse `GlobalData.packetSize`), werden diese Teilergebnisse zum Simulationsprogramm gesendet. Dazu hat jeder Thread eine Klasse `SendResult`, die mit der Funktion

```
- public boolean sendPacket (Object resultVector)
```

das Versenden von Objekten, an einen im Konstruktor der Klasse `SendResult` angegebenen Host, ermöglicht.

2.1.3 Die Kommunikation mit dem Simulationsprogramm

Es sind zwei verschiedene Arten der Kommunikation mit dem Simulationsprogramm zu unterscheiden: Die Simulationsdaten-Kommunikation (das Empfangen von Fahrzeugdaten und Senden von Ergebnisdaten), sowie die Kontroll-Kommunikation (das Senden und Empfangen von Kontroll- und Steuerungskommandos).

Die Kontroll-Kommunikation

Unmittelbar nach dem Start des Berechungsservers wird ein Thread gestartet (Klasse `ControlThread`), der die Routinen zur Kontroll-Kommunikation mit dem Simulationsprogramm bereitstellt:

```
/* thread to interact with requesting host                              */
control = new ControlThread (this);
control.start ();
```

Die `run()`-Methode des `ControlThreads` läuft dabei in einer endlosen Schleife, in der auf ServerSocket-Port 6677 (definiert in `GlobalData.ControlPORT`) auf Kontrollverbindungs-Anfragen gelauscht wird. Geht eine Verbindungsanfrage ein, wird ein Socket erstellt, über den in einer Schleife solange Kontrollnachrichten ausgetauscht werden können, bis das verbundene Simulationsprogramm die Kontrollverbindung schließt, oder die Verbindung aus anderen Gründen abgebrochen wird. Danach kehrt der Kontrollthread durch die Endlosschleife wieder in einen auf ServerSocket-Port 6677 lauschenden Zustand zurück. Der Ablauf der Kommunikation ist in Abbildung 2 dargestellt:

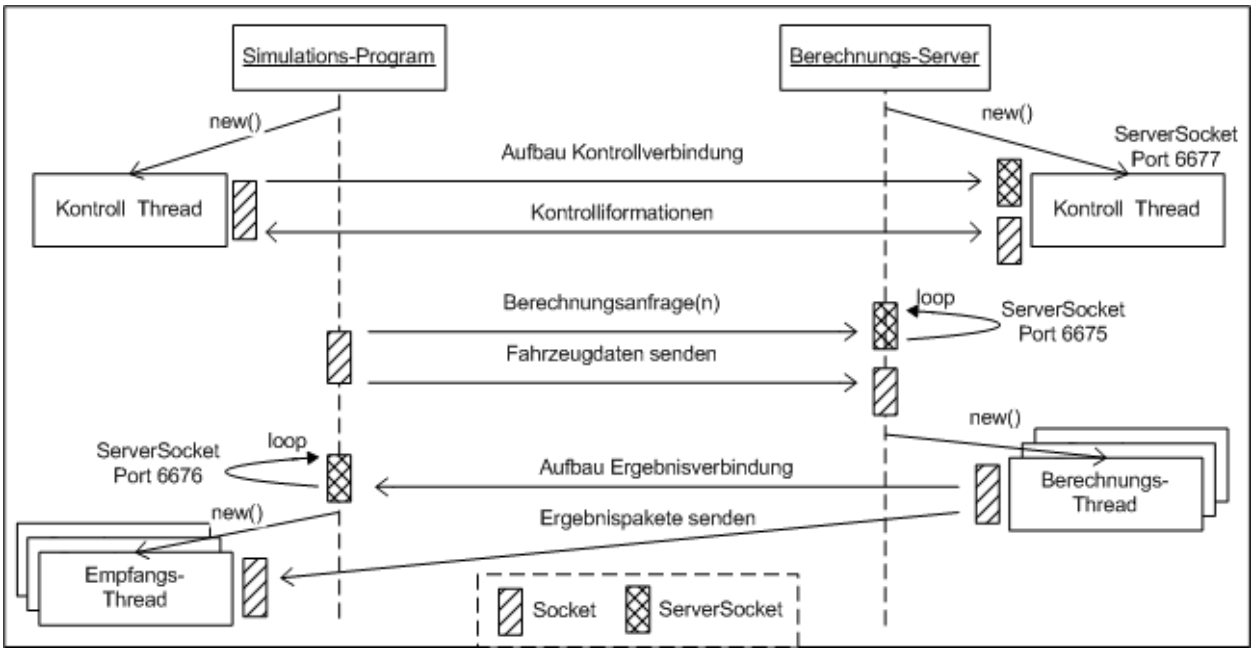


Abbildung 2: Kommunikationsabläufe

Die Kontrollsteuerung läuft dabei auf Basis der Nachrichtenübermittlung ab, wobei folgende Kontrollfunktionalitäten durch das Versenden von bestimmten Zeichenketten aufgerufen werden können:

Verfügbarkeit des Berechungsservers testen

Ankommende Zeichenfolge : `TEST_AVAILABILITY`

Antwort des Berechnungsservers: `IS_AVAILABLE`

Berechungen abbrechen:

Ankommende Zeichenfolge : `STOP_CALCULATION`

Antwort des Berechnungsservers: `SUCCESSFUL`

Aktion des Berechungsservers: `calcServ.stopCalculation ();`

Hauptprogramm ruft daraufhin `calculation[i].terminate ()` eines jeden Berechungsthreads auf. Diese Methode unterbricht (interrupt) und beendet die jeweilige run()-Funktion der Threads und führt somit zu einem Beenden der Berechnungsthreads.

Berechungsserver herunterfahren:

Ankommende Zeichenfolge : `TERMINATE`

Antwort des Berechnungsservers: `SUCCESSFUL`

Aktion des Berechungsservers: Herunterfahren aller Berechungsthreads und des gesamten Servers.

Der Kontrollthread horcht dabei auf dem Inputstream der Socketverbindung. Kommt eine Message an, überprüft er, ob die empfangene Zeichenkette als registriertes Kommando zur Verfügung steht und führt dieses Kommando gegebenen Falls aus.

Simulationsdaten-Kommunikation

Wurde der Berechnungsserver gestartet, lauscht im Hauptprogramm (Klasse `CalculationServer`) ein ServerSocket auf Port 6675 (definiert in `GlobalData.mainPort`). Dieser Port wartet auf Berechungsverbindungsanfragen des Simulationsprogramms. Geht eine Verbindungsanfrage auf diesem ServerSocket ein, wird ein Socket erstellt, über den die Fahrzeugdaten entgegengenommen werden, für die dann jeweils ein Berechnungsthread gestartet wird (siehe Punkt 2.1.2). Nach dem E inlesen der Fahrzeugdaten, kehrt der Server sofort wieder in einen auf Berechungsanfragen lauschenden Zustand über (Port 6675), während die Berechungsthreads im Hintergrund die Simulation berechnen, was folgender Programmausschnitt verdeutlicht:

```
for (;;)
{
  try
  {
    socket = serverSocket.accept ();
  }
  catch (IOException e)
  {
    System.err.println ("Couldn't initialize  Socket! " + e);
  }
  //... Fahrzeugdaten lesen, Verbindung abbauen
  //... Berechungsthreads für jedes Fahrzeug starten
}
```

Die Berechungsthreads übernehmen jeweils die Kommunikation mit dem Simulationsprogramm, um Ergebnisdaten zu senden. Dazu wird im Konstruktor jedes Berechungsthreads ein Objekt der Klasse `SendResult` erzeugt, dem die IP-Adresse des anfragenden Simulationscomputers übergeben wird:

```
sendResult = new SendResult (host);
```

`SendResult` baut dann eine Socketverbindung zum Simulationsprogramm auf (Verbindungsanfrage auf ServerSocket Port 6676 des Simulationsprogramms, definiert in `GlobalData.ResultPort`).

Über die Funktion `public boolean sendPacket (Object resultVector)` der Klasse `SendResult` kann der jeweilige Berechnungsthread über diese Verbindung Teilergebnisse (Vektor aus `SimResults`) während der Berechung zum Simulationsserver übermitteln. Der boolsche Rückgabewert dieser Funktion dient zum Beenden eines Berechnungsthreads, falls beim Senden eine Ergebnispaketes ein Fehler aufgetreten ist (z.B. Verbindung abgebrochen durch Simulationsprogramm). Der Ablauf der Simulationsdatenkommunikation ist ebenfalls in Abbildung 2 dargestellt.

2.2 Implementierung der Realtime-Schaubilder

Die Realtime-Schaubilder erfüllen zwei Aufgaben in dem Simulationsprogramm:

- Visualisierung der Simulations-Ergebniskommunikation mit Berechnungsserver
- Darstellung der Simulationsergebniswerte (Realtimeplotting)

Die Schaubilder sind in den Klassen `Graph, GraphPlotter` und `MultiGraphPlotter` implementiert, der Zusammenhang wird in Abbildung 3 dargestellt.

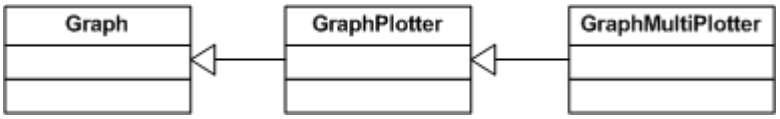


Abbildung 3: Schaubilder Klassen

Den Schaubildern steht im Simulationsprogramm ein eigenes Fenster (Klasse `GraphWindow`) zur Verfügung. In `GraphWindow` sind fünf verschiedene Schaubilder in drei Kategorien eines `JTabbedPane` (Registerkarten) untergebracht: Die erste Registerkarte „Drehzahl/Geschwindigkeit“ enthält zwei Objekt der Klasse `GraphPlotter` um die Geschwindigkeit sowie die Drehzahl darzustellen. Die zweite Registerkarte „Wirkende

Kräfte“ enthält ein Objekt der Klasse `GraphPlotter` um die Antriebskraft darzustellen und ein Objekt der Klasse `GraphMultiPlotter` (Darstellung zweier Funktionen in einem Schaubild) in dem die Luftreibungskraft und die Rollreibungskraft dargestellt werden. Im dritten Register wird der zurückgelegte Weg mit einem Objekt `GraphPlotter` gezeichnet.

Die Hintergrund Gitterstruktur wird bereits beim Sichtbarwerden der Registerkarten gezeichnet. Dies übernimmt die Funktion

- `public void drawGrid (Graphics bufferGraphics)`

die in den Klassen `GraphPlotter` und `GraphMultiPlotter` von `paint (Graphics g)` aufgerufen werden.

2.2.1 Visualisierung der Ergebniskommunikation

Treffen im Simulationsprogramm Teilergebnis-Vektoren ein, werden diese an die Klasse `GraphWindow` mit der Funktion

- `public void sendPackage (SerVector serVect)` weitergegeben. In `GraphWindow` werden die Teilergebnisse dann über die Funktion `private void drawAddedResult()` an die Schaubilder (`GraphPlotter, GraphMultiPlotter`) weitergegeben, wo sie mit der Funktion

- `public void addResults (Vector result)` dem lokalen Ergebnisvektor hinzugefügt werden. Die `paint()`-Funktionen der Schaubilder zeichnen dann in einer Schleife für alle Werte des lokalen Ergebnisvektors die dazugehörende Schaublidkurve. Die Schaubildkurve wird durch das Zeichnen einer Linie von Ergebnispunkt zu Ergebnispunkt realisiert. Das letzte Ergebnispaket enthält an letzter Stelle ein Endeflag (-1). `paint()` ruft bei Erkennen dieses Endeflags die Funktion `public void refresh()` auf, wodurch bei der nächsten Ergebnisvisualisierung, die Schaubilder neu gezeichnet werden können.

2.2.2 Realtimeplotting der Simulationsergebnisse

Sind alle Ergebnisse der Berechnung des Berechnungsservers eingetroffen, kann die Realtimeplotting-Funktion der Schaubilder genutzt werden. Dazu wird an das Schaubild-Fenster der komplette Ergebnisvektor einer Fahrzeugberechnung mit der Funktion

- `public void plotRealtime (SimulationResults simVect)`

übergeben. Diese Funktion übergibt die entsprechenden Simulationsergebnisse durch die Funktion `public void plotRealtime (Vector result)` der Schaubilder-Klassen an die Schaubilder. Danach wird ein Timer gestartet, der nach dem Ablauf der Zeit einer Berechnungsiteration den Impuls `plotImpulse ()` an die Schaubilder gibt, den nächsten Ergebniswert des Ergebnisvektors zu zeichnen, was im folgenden Programmausschnitt dargestellt wird:

```
timer = new Timer ((int) (GlobalData.deltaT * 1000), new ActionListener ()
{
  public void actionPerformed (ActionEvent e)
  {
    if (cntPoint < rpmVec.size ())
    {
      rpmGraph.plotImpulse();
      //Impulse an andere Schaubiler
      //...
    }
    //...
  }
//...
```

Abbildung 4 zeigt eine Ausgabe eins Schaubildes das von Schaubild der Klasse `GraphPlotter` gezeichnet wurde.

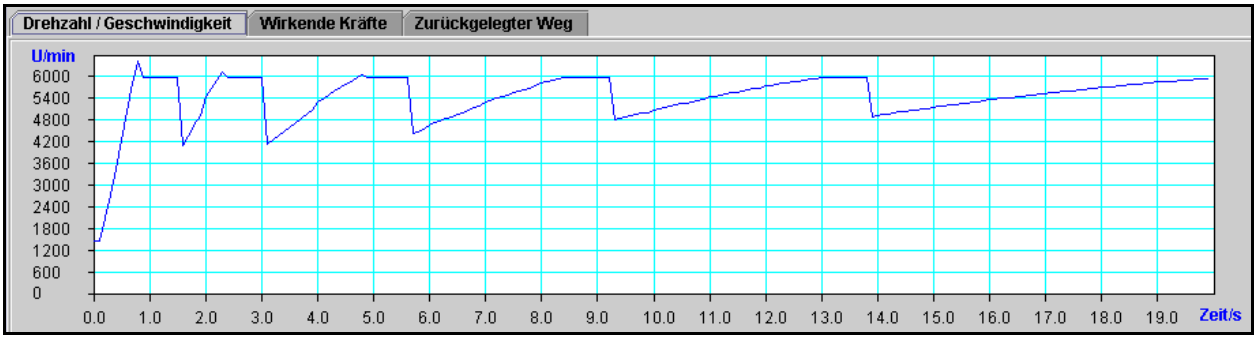


Abbildung 4: Screenshot eines Realtime Schaubildes

2.3 Implementierung der 3D Fahrzeug-Armaturen

Die Fahrzeugarmaturen wurden alle mithilfe von Standard Java3D Komponenten realisiert. Es wurde eine reine Programmierung der Armaturenkomponenten, ohne Verwendung von graphischen Hilfsprogrammen vorgesehen, da sich das Aussehen der Armaturen durch den Benutzer verändern lassen soll (siehe dazu Benutzerdokumentation Punkt 4). So sollen z.B. die Anzahl der Markierungsstriche, die

die Geschwindigkeit oder die Drehzahl markieren und deren Beschriftung vom Benutzer einstellbar sein. Dadurch ergibt sich, je nach Benutzerauswahl der Maximalgeschwindigkeit oder -drehzahl, eine Neuberechnung der Positionen dieser Elemente. Dies wäre, bei einer durch ein Grafikprogramm oder Modellierer gewonnene Objektdatei, bestehend aus den einzelnen Punkten nicht möglich (es sei denn, man würde für jede Einstellung, einen eigenen Tacho/Drehzahlmesser modellieren). Das gesamte Armaturenbrett wird als eine Transformgruppe in die Gesamtszene der Fahrzeugbeschleunigungs-Szene übernommen (siehe dazu Punkt ???). Abbildung 5 zeigt einen Screenshot des Armaturenbretts.

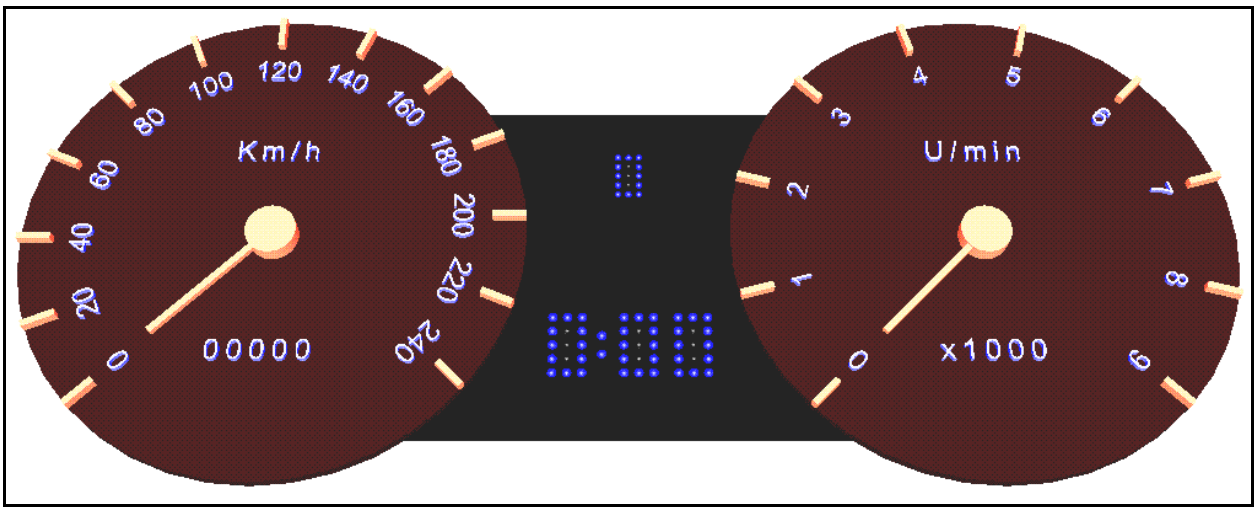


Abbildung 5: Das Armaturenbrett

2.3.1 Die Zusammensetzung der 3D-Armaturen

Die Zusammensetzung der einzelnen Komponenten des Armaturenbretts findet in der Klasse `DashBoard` satt. In dieser Klasse werden für alle Teilobjekte des Armaturenbretts, jeweils eine eigene Transformgruppe erstellt. Das Armaturenbrett besteht aus folgenden Teileobjekten:

- Tachometer mit Anzeige des zurückgelegten Weges
- Drehzahlmesser
- Stoppuhr (digitales Display mit Minuten-, 10 Sekunden- und Sekunden-Digit)
- Gangzähler (digital Display)
- Armaturenhintergrund

Mit der Funktion `createXY()` eines jeden Teilobjektes XY, werden diese dann den einzelnen Teilobjekt-Transformgruppen hinzugefügt. In den einzelnen Teilobjekt-Transformgruppen werden durch Translation die Objekte dann an die richtige Stelle im Armaturenbrett positioniert:

```
moveTacho.setTranslation (new Vector3f (-1.4f * cylRad, 0.0f, 0.0f));
```

Alle diese Teilobjekt-Transformgruppen werden der Transformgruppe `TGinclinate` hinzugefügt. `TGinclinate` ist dafür zuständig, das gesamte Armaturenbrett zu kippen (`inclinate.rotX (inclination)`), um eine leicht schräge Draufsicht auf die Armaturen zu ermöglichen. `TGinclinate` wird schließlich der übergeordneten Transformgruppe `TGDashBoard` hinzugefügt. Das gesamte Armaturenbrett kann somit durch die Methode `public TransformGroup createSceneGraph ()` der Klasse `DashBoard` in die Simulationsszene integriert werden.

Alle Positionen, sowie alle Größen eines jeden Elements im Armaturenbrett hängen von einer einzigen Größe ab: dem Radius des Tachohintergrund-Cylinders. Will man die Gesamtgröße des Armaturenbretts verändern, muss man lediglich den Wert `float cylRad` in der Klasse `DashBoard` verändern, alle anderen Größen werden im Verhältnis dieses Parameters berechnet.

Der obersten Transformgruppe `TGDashBoard` des Armaturenbretts ist noch ein direktionales Licht beigefügt, um durch Schattenwurf die Räumlichkeit der Armaturen zu verdeutlichen. Die Zusammensetzung der Transformgruppen und sämtlicher Teilobjekte des Armaturenbretts wird in Abbildung 6 dargestellt.

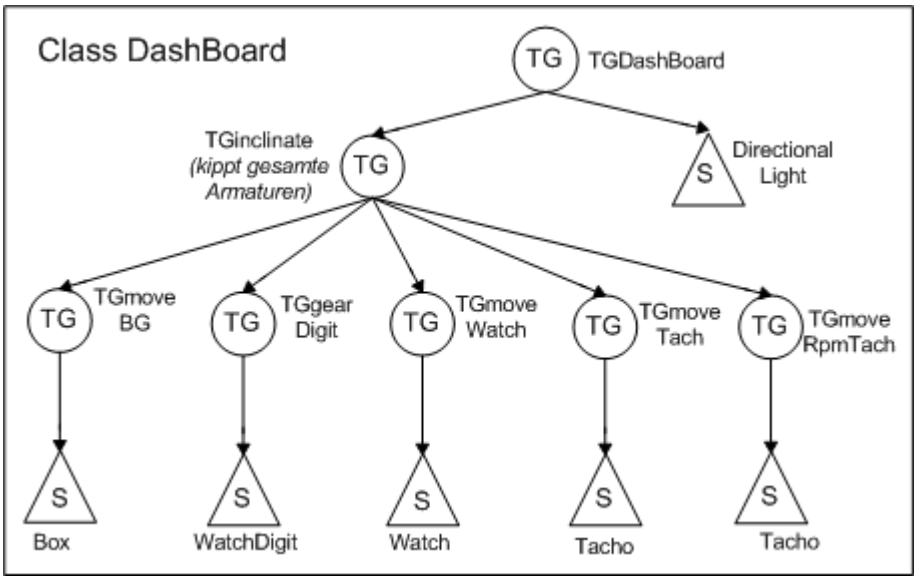


Abbildung 6: Zusammensetzung des Armaturenbretts

Im Folgenden wird auf den Aufbau der Teilkomponenten des Armaturenbretts eingegangen.

2.3.2 Die Tachometer (Geschwindigkeit und Drehzahl)

Der Geschwindigkeitstachometer und der Drehzahlmesser werden beide als Objekt der Klasse `Tacho` erzeugt. Ein Tacho ist aus verschiedenen primitiven Java3D Komponenten zusammengesetzt. Der Hintergrund der Tachos besteht aus einem Cylinder. Auf diesen Cylinder wurden in Abhängigkeit der Anzahl der Beschriftungselemente (abhängig von der Benutzerauswahl des Maximalwertes des Tachos) kreisförmig Java3D Boxen und Beschriftungen für die Bezeichnung der Boxen platziert. Die Verteilung der Boxen erfolgt dabei in einer Schleife. Die Positionen werden nach dem im folgenden Programmausschnitt mathematischen Prinzip berechnet:

```
for (i = 0; i <= boxNumber * 3 / 4; i++)
{
  box[i] = new Box (boxX, boxY, boxZ, app_box);
  float xPos = (float) (boxRadi * Math.cos ((360 / boxNumber) * i
        * Math.PI / 180));
  float yPos = (float) (boxRadi * Math.sin ((360 / boxNumber) * i
        * Math.PI / 180));
  // rotate boxes
  rotateBox.rotZ (-((i) * (2 * Math.PI) / (boxNumber)));
  // Move boxes
  rotateBox.setTranslation (new Vector3f (xPos, -yPos, cylHeight/2 + boxZ));

  // Beschriftung hinzufügen
  // ........
}
```

Die Schleife läuft dabei nur über 75% der Anzahl der Beschriftungsstriche (`boxNumber`), da im unteren Bereich des Tachos, zwischen Null und Maximalwert keine Beschriftungsstriche platziert werden sollen. Die Positionen der Boxen/Beschriftungen werden durch die trigonometrische Berechnung der `xPos` und `yPos` Koordinaten berechnet. Die Boxen und Beschriftungen werden dann rotiert, damit die Boxen zum Tachomittelpunkt zeigen und die Ausrichtung der Beschriftungen rechtwinklig zur Tachomittelpunktrichtung ist, was durch die Funktion

```
rotateBox.rotZ (-((i) * (2 * Math.PI) / (boxNumber)))
```

erfolgt. In Abbildung 7 ist durch die gestrichelte Box, die Ausrichtung einer Box ohne die Rotation dargestellt. Damit die Box[i] zum Mittelpunkt zeigt, muss Box[i] um i x 2 p / BoxAnzahl gedreht werden.

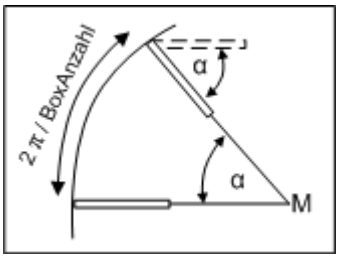


Abbildung 7: Drehen der Boxen/Tachobeschriftung

Der Tachozeiger wurde durch einen Cylinder und eine Box realisiert. Der Cylinder (`pointCyl`) stellt dabei die Halterung des Zeigers dar, die Box (`shaftBox`) den eigentlichen Zeiger. Der Zeiger wird durch eine Rotation in der Transformationsgruppe `TGmovPoint` so gedreht, dass er auf die Box mit der Beschriftung „0“ zeigt. Die Transformationsgruppe `TGrotPoint` ist für die Rotation des Zeigers in der Animation zuständig (siehe dazu Punkt 2.3.4). Neben der Beschriftung der Boxen und der Art des Tachos (Km/h oder U/Min), gibt es noch einen optionalen Counter (`textGeomCount`), der beim Geschwindigkeitstachometer für die Darstellung des zurückgelegten Weges genutzt wird. Der Wert dieses Counters wird in der Animation für jedes Berechnungsintervall auf den aktuellen Distanzwert gesetzt.

Alle Teilkomponenten des Tachos werden in der Group `GrTacho` zusammengefasst. Mit der Methode `public Group createTacho ()` der Klasse `Tacho` kann ein komplettes Tachoobjekt in das Armaturenbrett integriert werden. In Abbildung 8 ist die Zusammensetzung eines Tachos, dessen Teilobjekte und Transformgruppen dargestellt.

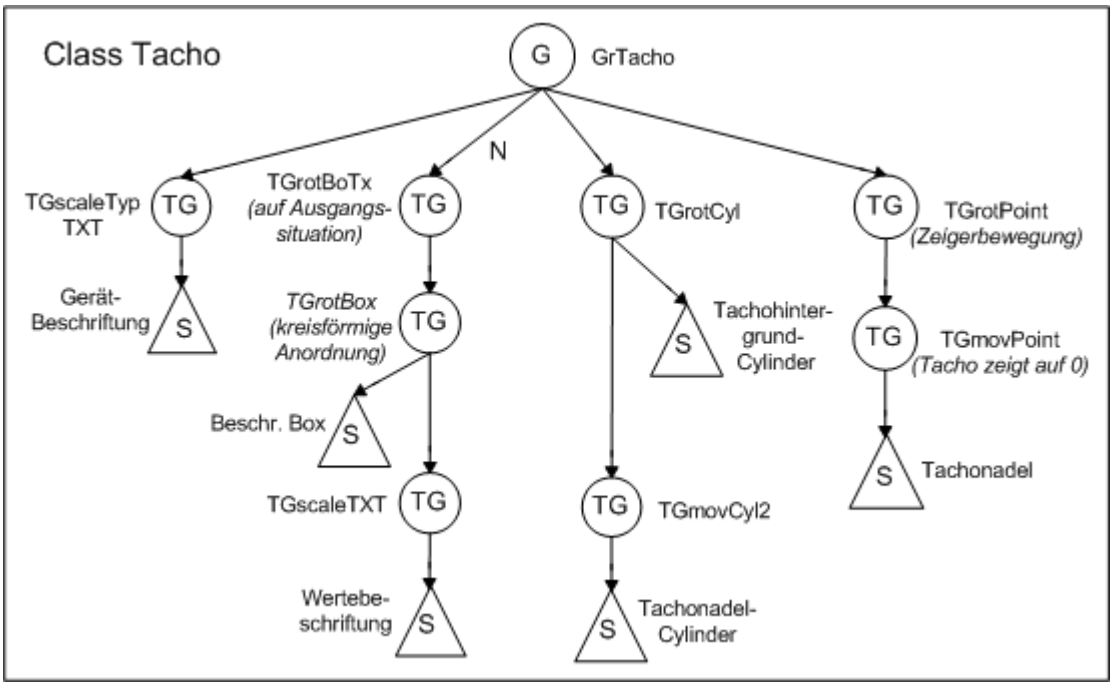


Abbildung 8: Zusammensetzung der Tachometer

2.3.3 Die Displays (Stoppuhr und Gangzähler)

Im Armaturenbrett sind neben den Tachometern zusätzlich Displays installiert. Ein Display dient als Stoppuhr zum Anzeigen der vergangenen Zeit seit Animationsbeginn. Ein zweites Display zeigt den momentan eingelegten Gang an. Die Stoppuhr ist in der Klasse `Watch` implementiert. `Watch` ist aus primitiven Java3D-Komponenten sowie Instanzen der Klasse `WatchDigit` zusammengesetzt. Die Stoppuhr besteht aus drei digitalen Zahlen, die die Minuten, 10xSekunden und Sekunden darstellen. Jede Digitale Zahl ist als eine Instanz von `WatchDigit` realisiert, auf die unten näher eingegangen wird. Das Minutendisplay ist durch zwei primitive Java3D Komponenten `Sphere[] sphereDot` getrennt, die in der Transformgruppe `TGmoveDots` hinzugefügt und dort positioniert werden. Die einzelnen d igitalen Zahlen werden den Transformgruppen `TGmoveWatch_` hinzugefügt und dort positioniert, wobei das _ für die jeweilige Zahl steht (z.B. `TGmoveWatchMin`). Die Funktion `public Group createWatch ()` der Klasse `Watch` hat als Rückgabewert eine Group, wodurch sich die Komplette Stoppuhr mit allen darin enthaltenen Elementen in das Armaturenbrett integrieren lässt.

Die Digitalenzahlen

Die Digitalzahlen sind in der Klasse `WatchDigit` implementiert. Eine Digitalzahl besteht aus 15 Kugeln (`sphere = new Sphere[15]`) die so angeordnet sind, dass der optische

Eindruck einer LED-Anzeige entsteht. Abbildung 9 stellt die Anordnung der Spheres dar.

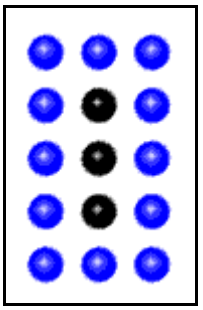


Abbildung 9: Aufbau einer Digitalzahl

Jede `sphere` wird in einer Schleife einer Transformgruppe `TGmoveSphere` zugeordnet, in der die Position festgelegt wird. Die `TGmoveSphere` Transformationsgruppen werden dann der Group `groupAddAll` hinzugefügt. Diese Group enthält alle Komponenten der Digitalzahl und kann über die Funktion `public Group createWatchDigit ()` in einer Szene integriert werden, wie z.B. in der oben beschrieben Stoppuhr. In Abbildung 10 ist der Zusammenhang der Gruppen und Objekte der Stoppuhr und der Digitalzahlen dargestellt.
Der Gangzähler der in der Klasse `DashBoad` dem Armaturenbrett hinzugefügt wird, ist ebenfalls eine Instanz des Klasse `WatchDigit`. Auf die Animation sämtlicher Armaturenkomponenten wird im folgenden Punkt eingegangen.

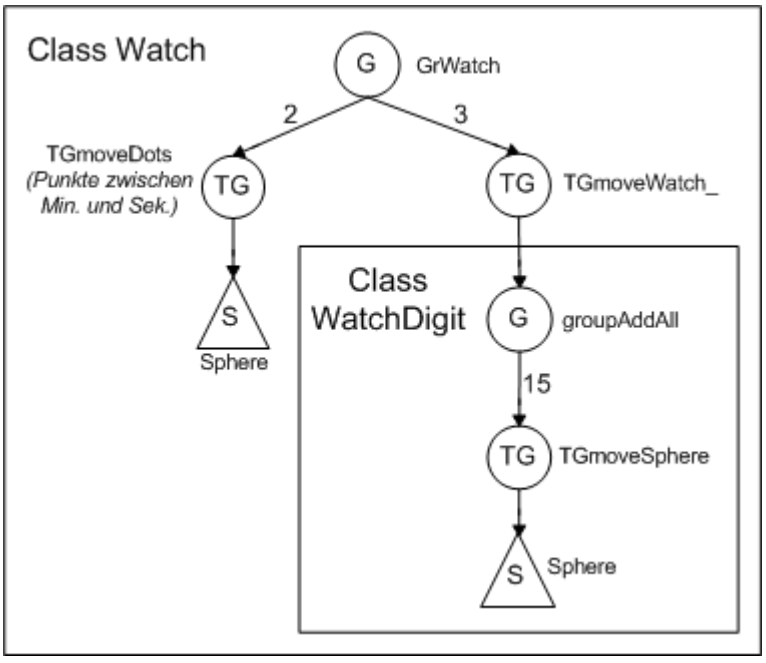


Abbildung 10: Zusammensetzung der Stoppuhr und Gangzähler

2.3.4 Die Animation der Armaturen

Im Armaturenbrett sind verschiedene Arten von „Animationen“ implementiert. Man darf diese Animationen jedoch nicht mit Java3D Animationen in Verbindung bringen, da die Animationen der der Java3D Szenen des Simulationsprogramms alle mit Java3D Behaviors implementiert wurden. Mit Behaviors ist ein genaueres Verhalten der Bewegung/Veränderung von Objekten möglich, da bei Java3D-Animationen lediglich in der Geschwindigkeit von einem Start zu einem Endpunkt interpoliert wird. Durch die Behaviors ist ein exaktes Verhalten der Armaturobjekte in Abhängigkeit der Simulationsergebniswerte möglich. Folgende „Animationen“ sind im Armaturenbrett implementiert:

Bewegung der Tachozeiger synchron zur Geschwindigkeit
Bewegung des Drehzahlzeigers synchron zur Drehzahl
Laufender Distanzzähler synchron zum zurückgelegten Weg
Laufende Stoppuhr synchron zur verstrichenen Zeit
Anzeige des Ganges synchron zum aktuell eingelegten Gang

Die Änderung der Objekte Tachozeiger, Drehzahlzeigers und Distanzzähler, wird dabei in der von `Behavior` abgeleiteten Klasse `TachoBehavior` vorgenommen. Die Stoppuhr und der Gangzähler in der von `Behavior` abgeleiteten Klasse `WatchBehavior`. In diesen Klassen gibt es zwei Arten von Wakeup Conditions:

`-WakeupOnElapsedTime(sleeptime)` wird verwendet, um für sämtliche Animationskomponenten alle `#sleeptime` Sekunden (für jede Berechungsiteration) die aktuellen Werte aus dem Ergebnisvektor zu lesen und die Komponenten entsprechend zu verändern.

`-WakeupOnBehaviorPost(...)` wird verwendet, um die Animationskomponenten nach einem Animationsdurchlauf auf die ursprüngliche Ausgangssituation zurück zu setzten und um die Behavior zu starten.

Den Behaviorklassen werden jeweils die Transformationsgruppen, bzw. die Objekte die in Abhängigkeit der Ergebniswerte verändert werden sollen übergeben. Mit den Funktionen:

```
- public void setAnimationData (double valLabel[], double valCount[])
- public void setWatchBehaviorData (int[] gear)
```

können von Außen die Simulationsergebnisse die für die Animation der jeweiligen Komponente benötigt werden den Behaviors übergeben werden.

Animation des Tacho-/Drehzahlzeigers

Die Position des Zeigers (der Winkel) wird für jede Iteration anhand der Geschwindigkeit neu berechnet. `rotation.rotZ (angle)` dreht den Zeiger in die zum darstellbaren Wert `valLabel[i]` passende Ausrichtung, wobei i für die Nummer der Iteration steht. Eine Umdrehung von Null bis zur größten Ausrichtung valMax würde 2 p 0.75 entsprechen. Durch eine Verhältnisrechnung ergibt sich:

valMax / 2 p 0.75 = valLabel / x

x = valLabel 1.5 p / valMax

Folgender Programmausschnitt zeigt die entsprechenden Anweisungen der Zeigerrotation:

```
factor = (Math.PI * 1.5) / valMax;
//...
angle = -(valLabel[i] * factor);
rotation.rotZ (angle);
targetTG.setTransform (rotation);
```

Abbildung 11 veranschaulicht die Berechung der Rotation des Zeigers:

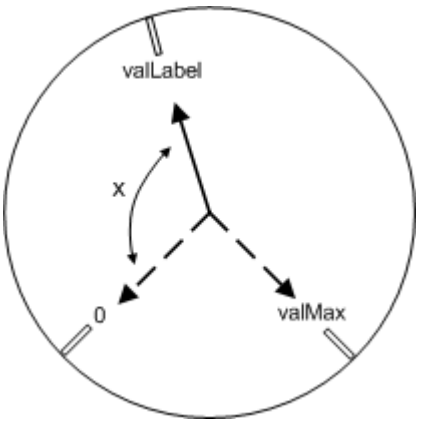


Abbildung 11: Rotation des Zeigers

Animation der Distanzanzeige (Kilometerzähler)

Für die Animation des Kilometerzählers wird der Klasse `TachoBehavior` das Text3D-Objekt `textGeomCount` übergeben. In `TachoBehavior` wird in jeder Iteration (durch

`WakeupOnElapsedTime(sleeptime)`) anhand der entsprechenden Simulationsergebnisse der String des `textGeomCount` –Objektes neu gesetzt. Je nach Länge der darzustellenden Zahl werden Nullen vorangestellt, damit der Kilometerzähler immer dieselbe Anzahl an Zahlen hat. Folgender Programmausschnitt zeigt die Neubelegung der Anzeige und das Voranstellen von Nullen:

```
if (valCount[i] < 10)
  textGeomCount.setString ("0000" + String.valueOf ((int) valCount[i]));
else if (valCount[i] < 100)
  textGeomCount.setString ("000" + String.valueOf ((int) valCount[i]));
//...
```

Animation der Stoppuhr und des Gangzähler

Der Klasse `WatchBehavior` werden im Konstruktor vier Arrays bestehend aus `Spheres` übergeben. Jeder Array stellt eine Digitalzahl, dar bestehend aus 15 Kugeln. Die erste Digitalzahl ist der Gangzahler, die folgenden drei stehen für die Stoppuhr. Zusätzlich werden im Konstruktor zwei Appearances mitgeliefert: Eine für eine leuchtende Sphere und eine für eine nicht leuchtende Sphere.

```
public WatchBehavior(Sphere[] sphereGear, Sphere[] sphereSec, Sphere[]
      sphere10Sec, Sphere[] sphereMin, Appearance app_sphere, Appearance
      app_sphereOff)
```

Die darzustellenden Werte werden für den Gangzähler aus dem Simulationsergebnis `gear[i]` für jede Iteration i entnommen. Der Wert der in der Stoppuhr dargestellt werden soll, ergibt sich aus der Anzahl der Iterationen. Liegt eine Änderung der darzustellenden Digitalzahl vor, wird der entsprechende Arrays aus Spheres und der darzustellende Wert `value` an die Funktion `private void changeDisplay (Sphere[] sphere, int value)` übergeben. Diese Funktion hat für jede Zahl zwischen 0 und 9 eine eigene Belegung der 15 Spheres, wobei je nach Zahl, den einzelnen Spheres leuchtende und nicht leuchtende Appearances zugeordnet werden. Der folgende Programmausschnitt zeigt die Zuordnung der Appearances zu den Spheres um den Wert „0“ in einem digitalen LED anzuzeigen:

```
private void changeDisplay (Sphere[] sphere, int value)
{
  if (value == 0)
  {
    sphere[0].setAppearance (app_sphere);
```

```
    sphere[1].setAppearance (app_sphere);
    sphere[2].setAppearance (app_sphere);
    sphere[3].setAppearance (app_sphere);
    sphere[4].setAppearance (app_sphere);
    sphere[5].setAppearance (app_sphere);
    sphere[7].setAppearance (app_sphereOff);
    sphere[9].setAppearance (app_sphere);
    sphere[10].setAppearance (app_sphere);
    sphere[11].setAppearance (app_sphere);
    sphere[12].setAppearance (app_sphere);
    sphere[13].setAppearance (app_sphere);
    sphere[14].setAppearance (app_sphere);
  }
  else if (value == 1)
  {
    //...
  }
  //else if ...
}
```

Für die Werte 1 bis 9 sind entsprechende Zuordnungen implementiert.

3 Projektdokumentation Teil 2

Im folgenden Dokumentationsteil werden die Programmteile dokumentiert, die von Thorsten Kohlhepp erstellt wurden.
Diese hier beschriebene Detaildokumentation bezieht sich ausschließlich auf das Hauptprogramm im Verzeichnis Window3D.
Die Benutzeroberfläche ist in Swing programmiert. Für die Animation wurde Java3D eingesetzt.

3.1 Grundlegendes

Die Anzeige des Hauptfensters wird in der Klasse `MainWindow` deklariert. Diese ist jedoch nur für die Ansicht der Menubar und deren Actionlistener zuständig.
Für die Darstellung der Ergebnissdaten entschieden wir uns für `JInternalFrames`. Dies erschwerte uns im Verlauf des Projekts zwar die Synchronisation der Fenster untereinander, ermöglichte uns aber unter anderem unabhängig voneinander zu arbeiten.
Die Nachrichtenweitergabe und die Steuerungskontrolle des Programms implementierten wir ebenfalls in einem `JInternalFrame`. Diese Klasse (`ControlWindow`) sammelt alle Ergebnissdaten und leitet sie weiter an die Fenster, die diese benötigen.
Dadurch garantieren wir, daß alle Pakete auch auf Seiten des Hauptprogramms gespeichert werden und dann auch angezeigt werden können.

3.2 Programmablauf

3.2.1 Ablauf des Hauptprogramms

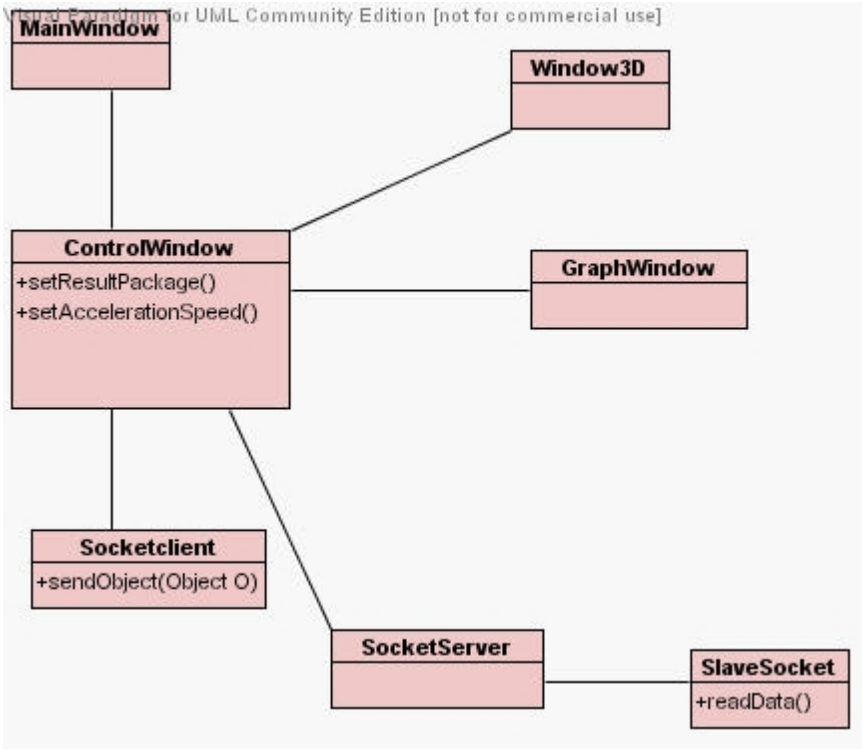


Abbildung 12: einfaches Klassendiagramm

Anhand dieses Klassendiagramms möchte ich den Ablauf des Programms erläutern. Ein vollständiges Klassendiagramm ist im Anhang eingefügt.

Die `ControlWindow`-Klasse speichert die geladenen Fahrzeugdateien in einem Vektor. Die Darstellung der Fahrzeugdateien im Kontrollfenster durch `JCheckBoxes` wird ebenfalls in einem weiteren Vektor gespeichtert. Dieser Vektor wird beim Starten der Berechnung durchlaufen und auf Selektion der `JCheckBoxes` überprüft. Ist eine `JCheckBox` selektiert, wird das Fahrzeug, das an der gleichen Stelle in dem Fahrzeugvektor enthalten ist, in einen temporären Vektor gespeichert. Der temporäre Vektor wird dann an den Berechnungsserver über die Klasse `Socketclient` mit der Methode `sendObject` übertragen. Nach Übertragung des Vektors wird die Verbindung abgebaut.

Schon beim Starten des Programms wurde ein Serversocket (Klasse: `SocketServer`) aufgebaut, der auf eingehende Verbindungen wartet. Hierüber werden die Ergebnisspakete vom B erechnungsserver empfangen. Trifft eine solche

Verbindungsanfrage ein wird ein neuer `SlaveSocket` aufgebaut, der die Pakete des Berechnungsservers empfängt und diese an die `ControlWindow` Klasse weiterleitet. Für jede Berechnung und deren Ergebnisspakete wird eine eigenständige Verbindung vom Berechnungsserver zum Hauptprogramm aufgebaut.
Das Ende der Pakete der Berechnung für ein Fahrzeug signalisiert der Berechnungsserver indem er im letzten Paket ein Endeflag setzt. Empfängt dieses das Hauptprogramm, übermittelt die Klasse `SlaveSocket` dies der `ControlWindow` Klasse und die Verbindung für dieses Fahrzeug wird abgebaut. Damit das Hauptprogramm erfährt, dass alle Fahrzeuge berechnet und empfangen wurden, wurde schon beim Versenden ein Zähler gesetzt, der die Anzahl der abgesendeten Fahrzeuge enthält. Beim Empfangen wird dann überprüft, wieviel neue Verbindungen vom Berechnungsserver aufgebaut wurden und ob jede Verbindung im letzten Paket, das Endeflag übermittelt bekommen hat. Ist dies der Fall sind alle Fahrzeuge berechnet worden und ebenfalls alle Pakete einer Berechnung empfangen worden. Da für jede Berechnung eines Fahrzeugs und deren Übermittlung der Ergebnissdaten eine eigentständige Verbindung aufgebaut wird, ist es möglich nach oben genannten Schema festzustellen, ob alle Daten vom Berechnungsserver empfangen wurden.
Die einzelnen Pakete werden sofort nachdem sie empfangen wurden an das entsprechende Graphenfenster (Klasse: `GraphWindow`) weitergereicht. Damit ist es möglich zu veranschaulichen wie schnell die Übertragung der Ergebnisse dauert. Denn die Zeit, die zwischen der Zeichnung der einzelnen Ergebnisspakete liegt beträgt genau

Berechnung + Übertragung + Zeichnen

Da aber die Berechnung und das Zeichnen der Ergebnissdaten auf Seiten des Hauptprogramms bei den heutigen Rechnergeschwindigkeiten vernachlässigt werden kann, kann man die Übertragungszeit mit der Zeit zwischen dem Zeichnen der einzelnen Ergebnisspakete gleichsetzen.
Natürlich sollte bei einer exakten Berechnung der Übertragungsgeschwindigkeit die Berechnung und das Zeichnen der Daten nicht vernachlässigt werden.
Außerdem bietet diese Variante der Übermittlung Parallelität in unserem Programm. Denn während der Berechnungsserver weiter die Beschleunigung des Fahrzeugs berrechnet, werden auf Seiten des Hauptprogramms die bereits berechneten Daten angezeigt. Nehmen wir dazu an, dass die Berechnung und die Anzeige auf unterschiedlichen Rechnern laufen, haben wir absolute Parallelität.

Nachdem alle Pakete empfangen wurden, werden sie im Hauptprogramm in einer `ArrayList`, die wiederum die komplette Berechnung eines Fahrzeugs in Vektoren enthält, gespeichert. Je nachdem welcher Radiobutton der Benutzer wählt, werden dann die Ergebnissdaten an die entsprechenden Fenster übermittelt. Das die Übergabe der Ergebnissdaten auch an das richtige Fenster erfolgt, wurde schon beim Absenden der Fahrzeuge die Titel der Fenster mit dem Namen des bzw. der Fahrzeuge gesetzt. Wenn die Ergebnisse vom Server empfangen werden, enthält jedes Paket den Namen des Fahrzeugs für das es bestimmt ist. Bei der Übergabe dann an die Fenster wird der Titel mit dem Namen das Pakets verglichen und bei Übereinstimmung an dieses übergeben. Da jedes Paket vom Berechnungsserver den Namen des Fahrzeugs enthält, ist somit auch keine Verwechslung möglich.
Bei einer erneuten Berechnung mit anderen Fahrzeugen werden auch die Titel der Fenster neu gesetzt. Somit ist auch auf Seiten der Fenster keine Verwechslung möglich.

3.2.2 Ablauf der Animation

Die Animation der Java3D Scene wurde mit `Behaviors` realisiert, da mit diesen eine genaue Zeitsteuerung der Animation möglich ist.
Sind alle Daten empfangen werden sie durch Drücken des „starte Realtime“ Buttons an die Klasse `Window3D` weitergereicht, die diese weiteleitet an die `Behaviors`. Dabei werden zunächst die Daten in den einzelnen `Behaviors` gesetzt. Dann wird Mithilfe der Methode `postId()` die `WatchBehavior` Klasse veranlasst ihre Arbeit durchzuführen. Die `WatchBehavior` Klasse stösst alle weiteren `Behaviors` an. Dies ermöglicht eine genauere Synchronisation der `Behaviors` untereinander. Alle `Behaviors` werden angewiesen ihre Operationen im 0,1 Sekunden Takt durchzuführen. Da die Operationen der Behaviors unterschiedlich lange Rechenzeiten in Anspruch nehmen, war es uns nicht möglich 100 % Synchronität der `Behaviors` untereinander zu erreichen. Dies wäre nur möglich gewesen, wenn jedes `Behavior` nach getaner Arbeit an einen übergeordneten Thread oder Prozess signalisiert hätte, dass die Arbeit verrichtet wurde. Der übergeordnete Prozess oder Thread müsste dann auf alle Endesignale der `Behaviors` warten und danach diese neu anstossen. Dies

wurde aus Zeitgründen jedoch nicht implementiert. Stattdessen wurde die Synchronität der `Behaviors` untereinander manuell abgestimmt, indem die Wartezeit der Durchläufe aufeinander abgestimmt wurde. So ändert nun die `TachoBehavior` und die `WatchBehavior` alle 0,11Sek. das Bild der Tachos bzw. der Mittelkonsole während die `AnimationBehavior` alle 0,1 Sek. das Bild der Umgebung verändert.

Bei der Animation der Umgebung bewegt sich nicht die Kamera durch die Szene sondern die Bäume, die Streifen auf der Strasse und die Pfosten bewegen sich auf den Betrachter zu. Wenn ein bestimmter Punkt erreicht ist, werden die Objekte wieder in ihren Ursprung zurückgesetzt und die Bewegung beginnt von neuem. Die so verwendete Vorgehensweise ermöglicht eine bessere Performance der Animation, da nicht die komplette Szene neu gerendert werden muss, sondern nur die Objekte, die sich bewegen.

3.3 Datenkapselung

3.3.1 Datenkapselung Fahrzeugdaten, Ergebnissdaten

Die Fahrzeuge werden nachdem sie geladen wurden in einem Objekt der Klasse `CarData` gespeichert. Diese einzelnen Objekte werden alle in einem Vektor der Klasse `SerVector` gespeichert. Ein temporärer Vektor des Typs `SerVector` enthält alle Fahrzeuge die selektiert wurden. Dieser Vektor wird an den Berechnungsserver gesendet.

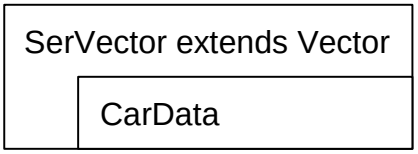


Abbildung 13: Datenkapselung der Fahrzeuge beim Übertragen an den Berechnungsserver

Empfangen werden wiederum Objekte vom Typ `SerVector` , jedoch enthält dieser jetzt im Regelfall 10 Objekte der Klasse `SimResult`. Das letzte Objekt, das übertragen wird, setzt außerdem in der Klasse `SimResult` das Endeflag, um das Ende der Übertragung zu signalisieren.

Diese Objekte von `SerVectors` werden, wenn sie sich auf das gleiche Fahrzeug beziehen in einem Objekt der Klasse `SimulationResults`, die wiederum einen Vektor darstellt, gespeichert. Die `SimulationResults`-Vektoren werden in einer `ArrayList` verwaltet.

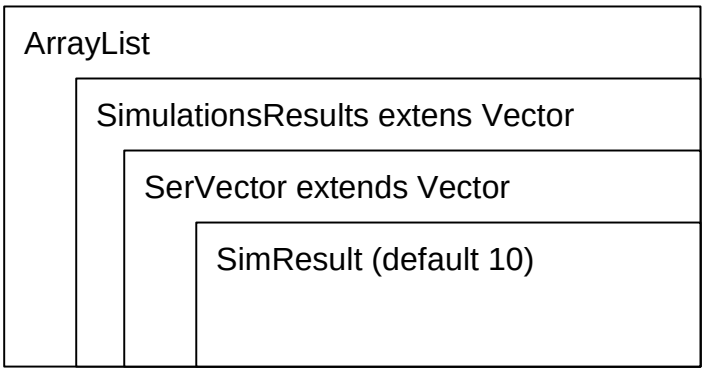


Abbildung 14: Datenkapselung der Ergebnissdaten

3.3.2 Datenkapselung Window3D Klasse

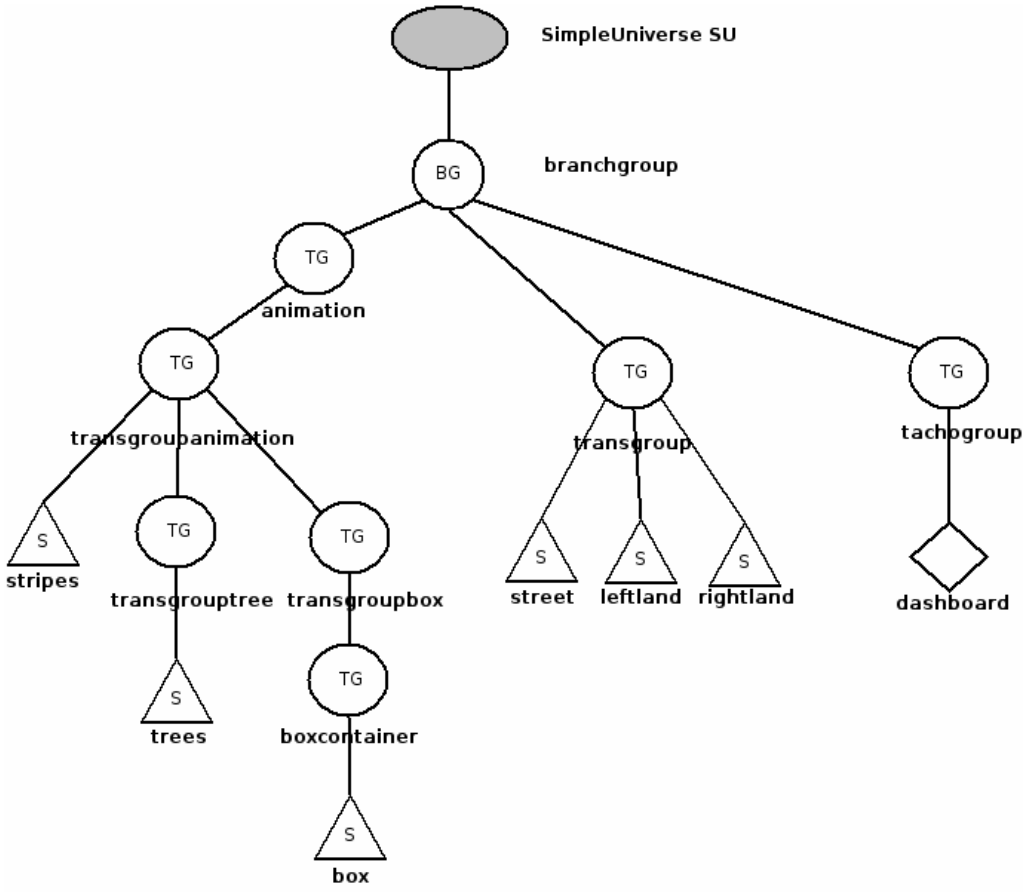


Abbildung 15: Szenen Graph

Oben gezeigte Abbildung zeigt die Datenkapselung in der Window3D Klasse. Dabei wurden alle Objekte die animiert werden sollen, in eine extra TransformGroup gepackt. Somit war es möglich nur die Objekte zu bewegen, die ein Fahren auf der Strasse simulieren. An der Abbildung können sie erkennen, daß wir nur die Streifen, die Bäume und die Pfosten bewegen lassen. Die Strasse, der rechte und der linke Grünstreifen bleiben dabei fix. Für das menschliche Auge ist dies jedoch kaum zu erkennen, daher haben wir auch keinen besonderen Qualitätsverlust der Animation.

3.4 Kontrollverbindung zu Berechnungsserver

Die Kontrollverbindung zum Berechnungsserver regelt die Klasse `RemoteWindow`, die ebenfalls durch die `MainWindow`-Klasse instantiert wird. In der Klasse `RemoteWindow` ist die Steuerung der Verbindung enthalten während die eigentliche Verbindung über die Klasse `ControlSocket` geregelt wird. Die Klasse `ControlSocket` hält dafür eine Schnittstelle bereit, um die Nachrichten zu setzen, die an den Berechnungsserver gesendet werden sollen. Dies ermöglicht die Methode `setSendMsg`. Da die Verbindung in einem Thread läuft, der wartet bis Nachrichten gesendet werden sollen, wird über die Methode `setSendMsg` ebenfalls ein `interrupt` ausgelöst, was das Warten unterbricht und veranlasst, dass die soeben neu gesetzte Nachricht an den Berechnungsserver gesendet wird. Nachdem die Nachricht übertragen wurde, kehrt der Socket sofort in den Lese-Modus und liest eine eingehende Nachricht. Diese wird dann über das Kontrollfenster ausgegeben und der Thread kehrt wieder in den Wartemodus. Die `ControlSocket` Klasse hält zu diesem Zweck eine Referenz auf die Klasse `RemoteWindow`. Über die Methode `setanswFieldMsg` ist es dem `ControlSocket` möglich die erhaltene Nachricht über das Kontrollfenster auszugeben.
Der `ControlSocket` empfängt vom Berechnungsserver entweder `IS_AVAILABLE` oder `SUCCESSFUL`. Je nachdem welche Anfrage vorher an den Berechnungsserver gesendet wurde, wird ein Flag gesetzt, dass dann die richtige Antwort überprüft. Wurde z. B. vorher nach Verfügbarkeit gefragt, wird das Flag auf 1 gesetzt. Antwortet der Berechnungsserver daraufhin mit `SUCCESSFUL` oder mit einem anderen beliebigen String als `IS_AVAILABLE`, ist der Berechnungsserver nicht verfügbar. Der Berechnungsserver muss mit `IS_AVAILABLE` antworten, um als verfügbar zu gelten. Bei den Anfragen `STOP_BERECHNUNG` oder `TERMINATE` antwortet der

Berechnungsserver mit `SUCCESSFUL`. Durch das vorher gesetzte Flag weiß der `ControlSocket` nun was erfolgreich durchgeführt wurde. Antwortet auch hier der Berechnungsserver mit einem anderen String als `SUCCESSFUL` ist die Anfrage fehlgeschlagen.

4 Benutzerdokumentation

In der folgenden Benutzerdokumentation wird die Installation und Benutzung des Simulationsprogramms, sowie des Berechnungsservers beschrieben.

4.1 Installation und Inbetriebnahme

Die Installation und Inbetriebnahme wird im Folgenden für Betriebssystem der Microsoft Windows Produktfamilie beschrieben.

4.1.1 Systemanforderungen

Für das voll funktionsfähige Programm benötigen sie mindestens JDK 1.4 mit java3d Unterstützung. Beides können sie unter www.java.sun.com kostenlos herunter laden. Installieren sie zunächst das JDK1.4.x . Wenn sie danach Java3d installieren, erhalten sie automatisch einen Dialog der ihnen bestätigt, daß das Java3d Paket für das JDK installiert wird.

4.1.2 Kompilieren der Programme

Wenn sie beides installiert haben passen sie ihre Pfade an. Gehen sie dabei wie folgt vor: Klicken sie mit der rechten Maustaste auf Arbeitsplatz und dann auf Eigenschaften. In dem folgenden Fenster wählen sie die Registerkarte „Erweitert“. In dieser klicken sie auf den Button „Umgebungsvariablen“.

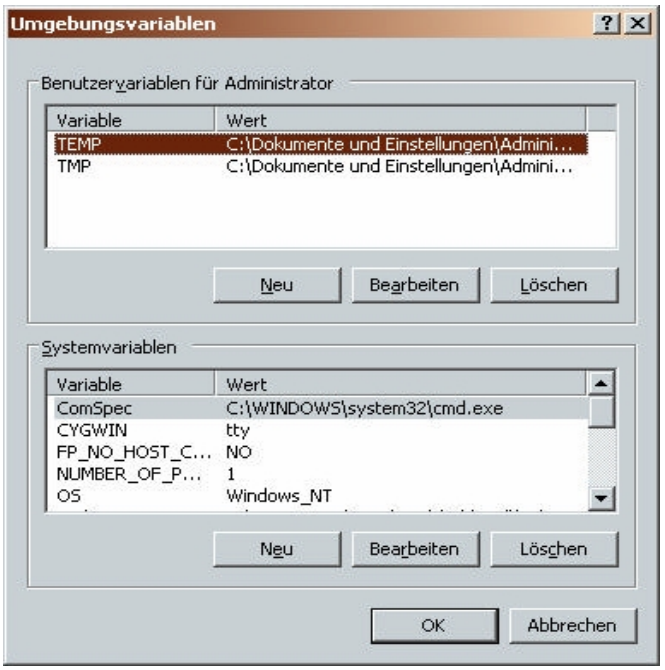


Abbildung 16: Umgebungsvariablen

Es öffnet sich ein weiterer Dialog in dem sie den Pfad für die Programme „javac“ und „java“ anpassen müssen.
In dem Fenster „Systemvariablen“ ist ein Eintrag „Path“ enthalten, h ier fügen sie am Ende den Pfad für die o. g. Programme an.

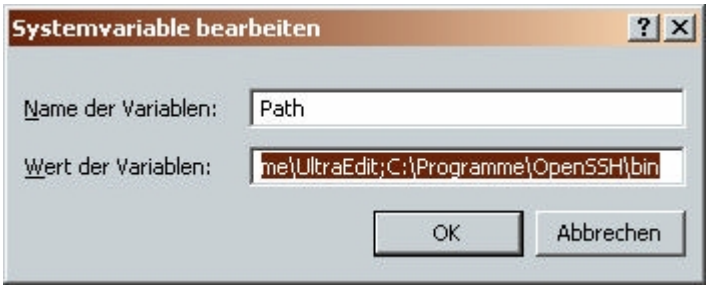


Abbildung 17: Einstellung des Pfades

Hinter dem letzten Eintrag fügen sie ein Semikolon an und tragen dann den Pfad ein. Standardmässig sollte dieser „C:\j2sdk1.4.1_02\bin“ lauten, ist bereits ein solcher Eintrag vorhanden müssen sie hier nichts weiter vornehmen.
Wenn sie den Eintrag vorgenommen haben, müssen sie das Programm zunächst kompilieren. Dies erledigen sie indem sie das Skript „compile.bat“ benutzen. Klicken sie hier einfach doppelt und das Programm wird kompiliert. Nachdem das Programm kompiliert ist, sollten in den Verzeichnissen Beschleunigung und Window3D mehrere Class-Dateien enthalten sein.

4.1.3 Ausführen der Programme

Wenn sie das Programm starten wollen, müssen sie zunächst etwas Grundlegendes zu diesem Programm wissen. Das Programm besteht aus 2 Teilen. Der erste Teil ist der Berechnungsserver und ist unter dem Verzeichnis Beschleunigung zu finden. Der zweite Teil ist das Fenster für die Darstellung der Berechnungsergebnisse und ist im Verzeichnis Window3D zu finden. Sie können beide Teile auf ein und demselben Rechner starten. Aber es ist auch möglich das Berechnungsprogramm auf einem entfernten Rechner zu starten. Dabei müssen sie nur beachten, dass die Firewall nicht die Ports 6677, 6675 und 6676 auf beiden Rechnern blockiert.
Sie können die Programme jeweils einzelnen mit den Skripten „ `runclient.bat` “ und „`runwindow.bat`“ starten.

4.2 Verbindungsaufbau zu einem Berechnungsserver

4.2.1 Kontrollfunktionen, Fernsteuerung des Berechnungsservers

Die Verbindung zum Berechnungsprogramm erfolgt automatisch, sofern der Berechungsserver und das Simulationsprogramm auf dem Selben Rechner ausgeführt werden, da standardmäßig mit der IP-Adresse 127.0.0.1 verbunden wird. Wollen Sie mit einem entfernten Rechner Simulationen berechnen lassen, müssen Sie erst eine Remote-Verbindung (Fernsteuerung) zu dem entfernten Berechungsserver aufbauen, was im Folgenden beschrieben wird.

Sie haben die Möglichkeit den Berechnungsserver fernzusteuern. In dem Hauptprogramm finden sie einen Menupunkt *Kontrolle*, welches ein Untermenü *Remote Verbindung* besitzt.

Klicken sie auf dieses erhalten sie einen Dialog mit dem sie den Berechnungsserver fernsteuern können. Dazu sollten sie sich zunächst mit dem Berechnungsserver verbinden. Dies können sie indem sie auf den Button „Verbinde“ klicken.

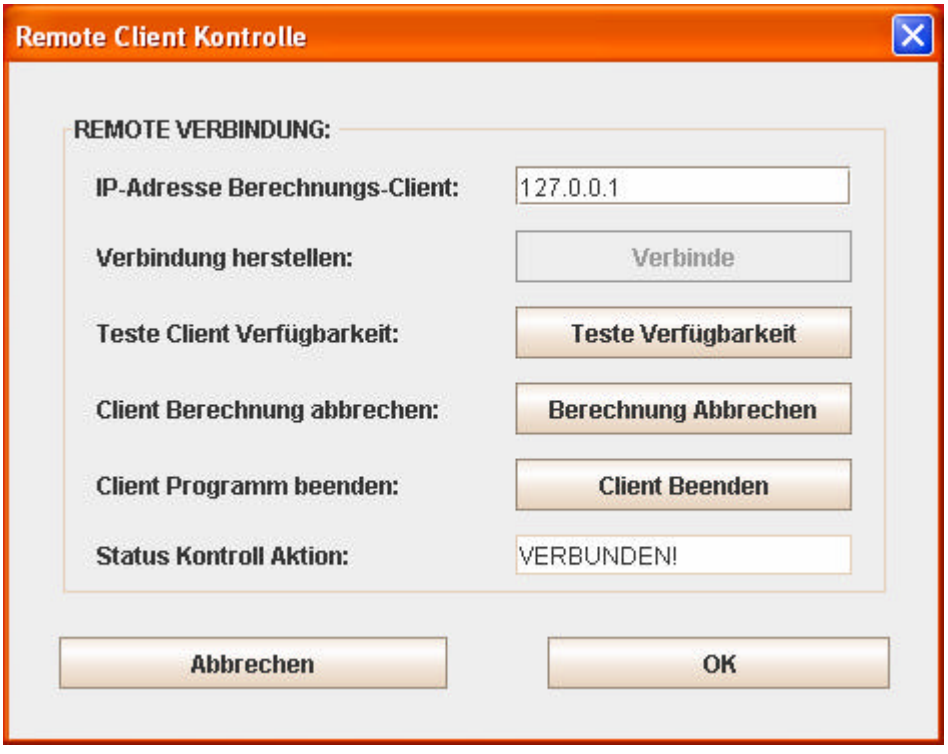


Abbildung 18: Optionen der Remote Verbindung

Natürlich sollte im Feld IP-Adresse, die des Berechnungsservers stehen, ansonsten erhalten sie die Fehlermeldung, dass die Verbindung zum Server fehlschlug. Wenn die Verbindung erfolgreich verlaufen ist, aktivieren sich automatisch, die zuvor deaktivierten Buttons. Beachten sie, wenn sie den Berechnungsserver anhalten, haben sie keine

Möglichkeit mehr von diesem Programm aus den Berechnungsserver neu zu starten, dies muss manuell geschehen.
Außerdem sollten sie wissen wenn sie einmal die Verbindung aufgebaut haben und dann den Dialog verlassen, wird die Verbindung zu dem Berechnungsserver nicht abgebaut. Sie können also immer wieder das Fenster aufrufen und die Verbindung bleibt bestehen.

4.3 Laden eines Fahrzeuges

4.3.1 Hinzufügen eines Fahrzeuges

Bei der Benutzung des Programms müssen sie beachten, daß sie nur bestimmte Dateitypen laden können. Außerdem müssen diese Dateien einen bestimmten Aufbau haben.
Sie können nur Dateien laden, die die Endung „.car“ besitzen. Diese Dateien können sie in jedem beliebigen Ordner ablegen. Sie müssen diese Dateien nicht in dem Programmpfad hinterlegen.
Bevor sie eine Berechnung durchführen können, müssen sie Fahrzeuge in das Programm laden. Gehen sie dabei wie folgt vor: Klicken sie auf *„Datei* à *Fahrzeug Laden*“. Es öffnet sich ein Dialogfenster indem sie ausschließlich Dateien auswählen können, die die Endung „.car“ tragen.

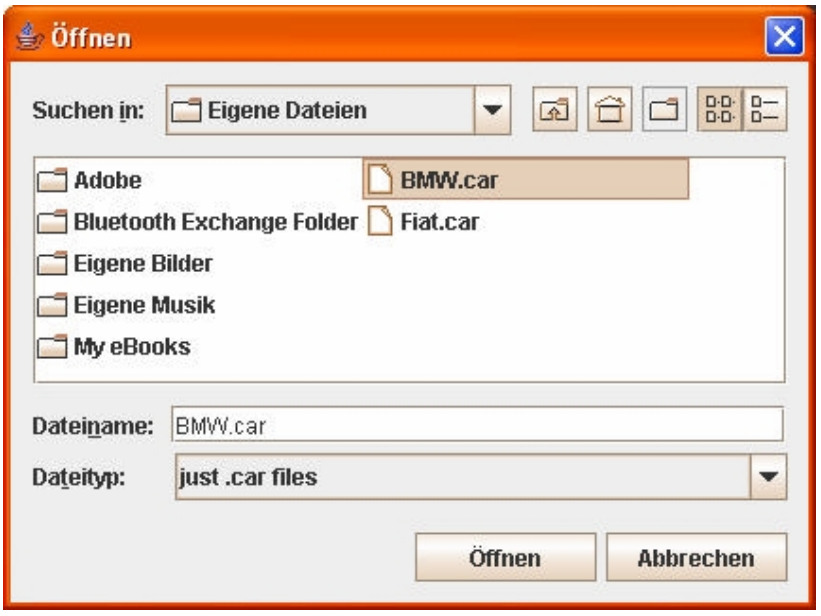


Abbildung 19: Hinzufügen eines Fahrzeugs

Wählen sie hier nun eine solche Datei aus und klicken sie anschließend auf *Öffnen*. Das Programm wird nach erfolgreichem Laden der Datei ein Fenster anzeigen, indem sie nochmals die Daten des Fahrzeugs angezeigt bekommen. Außerdem sollte sich im „Control“-Fenster eine Auswahlbox hinzugefügt haben. Wenn sie diese ausgewählt haben, wird diese mit Klick auf den Button „Starte Berechnung“ auch berechnet.

4.3.2 Aufbau der .car Dateien

Der Inhalt der Datei sollte folgenden Aufbau besitzen:
Am Anfang einer jeden Zeile steht der Variablenname, gefolgt von einem = und der Deklaration dieser Variablen. Abschließend muss jede Zeile ein Semikolon enthalten. Es ist unbedingt erforderlich, daß in jeder Zeile nur eine Variable deklariert wird. Mehrfachdeklaration kann zu Programmfehlern führen.
Exemplarischer Aufbau einer solchen Datei:

```
typ =Audi.car;
masse = 1500;
flaeche = 2.1;
cw = 0.3;
rollReibFaktor = 0.03;
leerDrehzahl = 1501;
schaltDrehzahl = 4800;
schaltZeit = 0.9;
reifRRadius = 0.23;
differential = 2.51;
drehMom[0] = 420.0;
drehMom[1] = 440.0;
drehMom[2] = 470.0;
drehMom[3] = 500.0;
drehMom[4] = 530.0;
drehMom[5] = 550.0;
drehMom[6] = 540.0;
drehMom[7] = 530.0;
drehMom[8] = 470.0;
drehMom[9] = 400.0;
drehz[0] = 1500;
drehz[1] = 2000;
drehz[2] = 2500;
drehz[3] = 3000;
drehz[4] = 3500;
drehz[5] = 4000;
drehz[6] = 4500;
drehz[7] = 5000;
drehz[8] = 5500;
drehz[9] = 6000;
gangzahl =6;
gang[0] = 4.254;
```

```
gang[1] = 2.534;
gang[2] = 1.682;
gang[3] = 1.235;
gang[4] = 1.000;
gang[5] = 0.831;
```

Die Variablennamen müssen unbedingt eingehalten werden, jedoch ist es möglich die Anzahl der Gänge, der Drehzahl oder die Drehmomente zu ändern. Sie sollten jedoch beachten, wenn sie die Anzahl der Gänge verändern, auch die Variable „ `gangzahl`“ dementsprechend abzuändern.
Weiterhin ist zu beachten, daß die Variable `typ` mit dem Namen der Datei übereinstimmt.

4.3.3 Ändern von Fahrzeugdaten

Sie haben außerdem die Möglichkeit die Daten des Fahrzeugs nach Laden in das Programm noch zu verändern. In dem Fenster, das sich nach Laden der Datei geöffnet hat, können sie in den entsprechenden Feldern Änderungen vornehmen. Nach den Änderungen sollten sie diese speichern mit Klick auf den Button „*Änderungen speichern*“. Die nun geänderten Daten werden auch so an den Berechnungsserver gesendet und es wird mit diesen Daten berechnet. Allerdings werden diese Änderungen nicht in die Datei geschrieben.

4.4 Berechungen starten

Die Berechnung starten sie indem sie auf den Button „*Starte Berechnung*“ klicken. Dabei müssen sie jedoch beachten, dass bereits mindestens ein Fahrzeug geladen wurde und einer der 3 Radiobuttons ausgewählt sein muss.
Mit den Radiobuttons wählen sie ihre Ansicht. Sie können dabei wählen ob sie nur Animation, nur Graphen oder beides bei einem Fahrzeug angezeigt bekommen möchten.

Wählen sie entweder „nur Graphen“ oder „nur Animationen“ können sie maximal 2 Fahrzeuge zur Berechnung senden. Dies wurde deshalb eingeschränkt, da es aus Gründen der Übersichtlichkeit nicht möglich ist, mehr als 2 Fenster anzeigen zu lassen. Sie können jedoch mehrere Fahrzeuge laden und diese unselektiert lassen.

Wenn sie die Berechnung gestartet haben, w ird das Eintreffen der Pakete in dem Graphen-Fenster, sofern sie dies ausgewählt haben, sofort angezeigt. In dem 3D-Animationsfenster wird das Eintreffen der Pakete vom Berechnungsserver nicht direkt angezeigt. Wenn alle Pakete eingetroffen sind aktiviert sich automatisch der Button für die Realtime Simulation, wodurch die Animation gestartet werden kann..

4.4.1 Gemischter Modus

Wenn sie den Radiobutton „*ein Fahrzeug*“ gewählt haben, bekommen sie den gemischten Modus angezeigt, was bedeutet, daß das Programm ihnen sowohl die Animation als auch die dazugehörenden Graphen anzeigt. Jedoch sollten sie hierbei beachten, daß sie nur ein Fahrzeug zur Berechnung senden können. Dies wird vom Programm automatisch gesteuert, indem alle Fahrzeuge bis auf eines deselektiert werden und diese auch nicht mehr selektiert werden können.

4.4.2 Graphen Modus

Wenn sie denn Radiobutton „nur Graphen“ ausgewählt haben, können sie maximal 2 Fahrzeuge zur Berechnung senden. Dies wird ebenfalls automatisch gesteuert, indem das Programm sie nicht mehr als 2 Fahrzeuge selektieren lässt. Senden sie nun 2 Fahrzeuge oder weniger zur Berechnung werden 2 bzw. ein Graphenfenster geöffnet. Der Titel der Fenster ist entsprechend der Fahrzeuge gesetzt. Die Daten werden immer in 10er Paketen vom Berechnungsserver gesendet. Sofort nach Eintreffen eines solchen Paketes, wird der Graph mit diesen Daten gezeichnet bzw. erweitert. Man kann dadurch genau mitverfolgen, wann Daten vom Berechnungsserver eintreffen. Nachdem alle Daten vom Berechnungsserver empfangen wurden, aktiviert sich der „starte Realtime“ Button. Dazu jedoch mehr unter 4.5.
Im Graphen-Fenster haben sie weiterhin die Möglichkeit, zwischen den einzelnen Tabs der Ergebnisewertdarstellungen zu wechseln.

4.4.3 Animationsmodus

Auch im Animationsmodus können sie maximal zwei Fahrzeuge zur Berechnung senden. Auch hier werden die Titel der Fenster entsprechend der zu berechnenden Fahrzeuge gesetzt. Jedoch werden hier nicht wie im Graphen Modus das Eintreffen der

Pakete angezeigt. Erst nach Eintreffen aller Pakete vom Berechnungsserver kann die Animation gestartet werden, was wiederum durch Aktivieren des „starte Realtime“ Buttons signalisiert wird.

4.5 Realtime-Animationen

Nachdem alle Ergebnisse der Berechnungsanfragen an den Berechnungsserver vollständig eingetroffen sind, besteht die Möglichkeit, sich die Ergebnisse in Realtime darstellen zu lassen. Das Eintreffen des letzten Ergebnispaketes einer Berechnung, wird durch das Aktivsetzen des „*Starte Realtime*“-Buttons signalisiert. Bevor man die Animation startet, besteht die Möglichkeit verschiedene Kombinationen an Ergebnisdarstellungen auszuwählen, die im Folgenden beschrieben werden:

4.5.1 Graphen Modus

Durch Auswahl der Radiobuttons *„Nur Graphen“* im Steuerungsfenster, gelangt man in den Graphen Modus. In diesem Modus werden die Ergebniswerte in Schaubildern in Realtime gezeichnet. Wurde nur ein Fahrzeug zur Berechung an den Berechungsserver gegeben, wird nur ein Fenster mit Schaubildern gezeichnet. Im anderen Fall, werden in zwei Fenster parallel Schaubilder gezeichnet und gegeneinander dargestellt. Während der Realtimedarstellung kann man aus den einzelnen Karteikarten die verschiedenen Ergebniswerte (Geschwindigkeit, Drehzahl, wirkende Kräfte, zurückgelegter Weg) darstellen lassen.

4.5.2 Animationsmodus

Durch Auswahl der Radiobuttons *„Nur Animation“* im Steuerungsfenster, gelangt man in den Animationsmodus. Im Animationsmodus werden lediglich die Fenster mit der 3D-Animation, mit Sicht auf das Armaturenbrett und Blick durch die Windschutzscheibe dargestellt. Wie im Graphenmodus wird, wenn nur ein Fahrzeug ausgewählt wurde, nur ein Fenster zur Animation dargestellt. Andernfalls werden in zwei 3D-Animationsfenster die Fahrzeuge im Vergleich dargestellt. Als Ergebniswerte werden dabei die Geschwindigkeit im linken Tachometer und Drehzahl im rechten Drehzahlmesser dargestellt. Zusätzlich kann der zurückgelegte Weg im Tachometer, der momentan eingelegte Gang im Gangzähler zwischen den Tacho, sowie die aktuelle verstrichene Zeit, seit Beginn des Animationsstarts unterhalb der Tachos abgelesen werden.

4.5.3 Gemischter Modus

Im gemischten Modus (Auswahl *„ein Fahrzeug“* im Steuerungsfenster) werden von einem Fahrzeug die Ergebniswerte in Realtime durch exakte Schaubilder, sowie durch die 3D-Animation dargestellt. Wurden die Berechnungen abgeschlossen, kann im Steuerungsfenster ein beliebiges Fahrzeug ausgewählt werden, um die Realtimdarstellung zu starten.
Grundsätzlich kann man, wenn die Ergebniswerte alle eingetroffen sind, wiederholt solange verschiedene Kombinationen von Realtimedarstellungen anzeigen lassen, bis eine neue Berechnung beim Berechnungsserver angefordert w ird. Will man die Eigenschaften der Darstellungselemente verändern (siehe Punkt 4.6), so gehen die Simulationsdaten nicht verloren, sondern können auch nach dem Neuaufbau der Fenster wieder dargestellt werden.

4.6 Darstellungseigenschaften ändern

Sollten die Obergrenzen der Darstellungselemente nicht ausreichen (z.B. falls der Beschleunigungsvorgang länger dauert, als die Schaubilder zeitlich darstellen lassen, oder die Geschwindigkeit, die Grenzen des aktuellen Tachos übersteigt) gibt es die Möglichkeit die Einstellungen dieser Elemente zu ändern. Die Änderungen kann man in der Menüleiste unter *Einstellungen, Graphen Einstellungen* und *3D Einstellungen* vornehmen.

4.6.1 Eigenschaften der Schaubilder ändern

Die Eigenschaften der Schaubilder lassen sich beliebig durch den Benutzer ändern. Für jede Achse, der dargestellten Schaubilder, lässt sich ein neuer Maximalwert eingeben. In Abbildung 20 sind die Standardwerte der Graphen dargestellt, die bei Programmstart eingestellt sind. Will man eine, oder mehrere Achsendimensionen ändern, muss man eine neue Obergrenze in das entsprechende Textfeld eingeben und mit *OK* bestätigen.

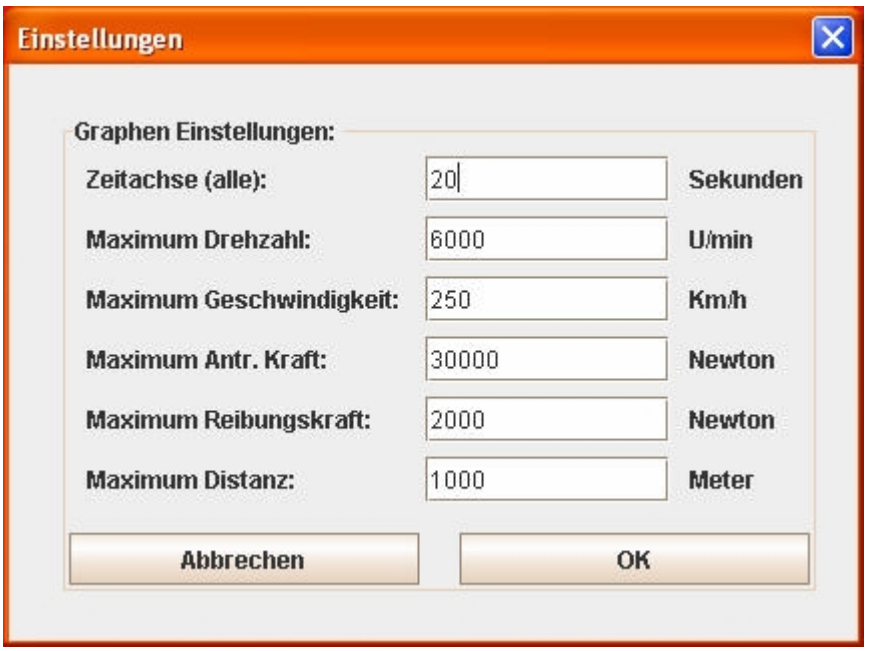


Abbildung 20: Graphen Einstellungen

4.6.2 Eigenschaften der 3D Animation ändern

Die Armaturen der 3D Fenster lassen sich auf ähnliche Weise einstellen. Unter dem Menüpunkt *Einstellungen,* gelangt man unter *3D Einstellungen* zu einem Auswahldialog, in dem man die gewünschten Parameter auswählen kann (maximale Geschwindigkeit des Tachos und maximale Drehzahkl des Drehzahlmessers). Um falsche Benutzeingaben zu vermeiden, kann man hierbei aus vordefinierten Armatureigenschaften wählen. Per Default sind die Eigenschaften, die in Abbildung 21 dargestellt sind aktiviert.

Abbildung 21: Armaturen Einstellungen

Die Änderungen der Einstellungen werden bei den Schaubildern, sowie bei den 3D Darstellungen sofort nach Bestätigen des OK-Buttons aktiv.

4.7 Die Bedienung des Berechnungs-Servers

Da das Berechnungsprogramm nur als Berechnungsunterstützung gedacht ist und nur als Berechnungsserver verteilt im Netz „im Hintergrund“ läuft, gibt es, nachdem der Server gestartet wurde, nicht viel Benutzerinteraktivität. Es steht jedoch nach Programmstart eine Konsole zur Verfügung, die Benutzerkommandos entgegen nimmt, die im Folgenden beschrieben werden:

4.7.1 Änderung der Sleeptime

Durch den Befehl „*sleeptime“* kann die Sleeptime des Berechnungsservers eingestellt werden. Nach Eingabe des Befehls, wird die aktuelle Sleeptime angezeigt (siehe Abbildung 22). Danach wird die neue Sleeptime von der Tastatur entgegen genommen. Bei ungültigen Eingaben erfolgt eine erneute Abfrage. Die Sleeptime kann jederzeit verändert werden, auch während laufenden Berechnungen. Änderungen werden sofort aktiv.

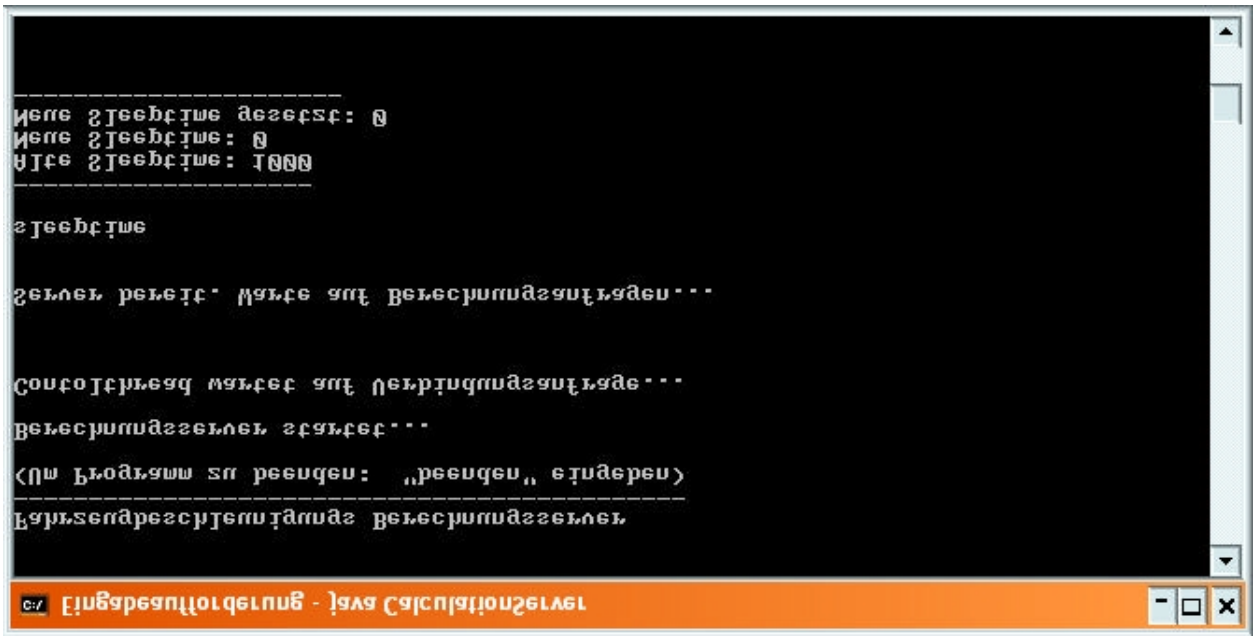


Abbildung 22: Änderung der Sleeptime

4.7.2 Beenden des Berechungsprogramms

Mit dem Befehl „*beenden“* kann der Berechnungsserver heruntergefahren werden. Bestehende Verbindungen werden sofort abgebrochen, ebenso laufende Berechnungen! Dieser Befehl kann ebenfalls zu jedem Zeitpunkt eingegeben werden und wird sofort ausgeführt.

5 Anhang:

5.1 Abbildungsverzeichnis:

Abbildung 1: Berechnungsmodell, Iterationskreislauf.....8
Abbildung 2: Kommunikationsabläufe.....10
Abbildung 3: Schaubilder Klassen.....12
Abbildung 4: Screenshot eines Realtime Schaubildes.....14
Abbildung 5: Das Armaturenbrett.....15
Abbildung 6: Zusammensetzung des Armaturenbretts.....16
Abbildung 7: Drehen der Boxen/Tachobeschriftung.....18
Abbildung 8: Zusammensetzung der Tachometer.....19
Abbildung 9: Aufbau einer Digitalzahl.....20
Abbildung 10: Zusammensetzung der Stoppuhr und Gangzähler.....20
Abbildung 11: Rotation des Zeigers.....22
Abbildung 12: einfaches Klassendiagramm.....26
Abbildung 13: Datenkapselung der Fahrzeuge.....29
Abbildung 14: Datenkapselung der Ergebnissdaten.....30
Abbildung 15: Szenen Graph.....30
Abbildung 16: Umgebungsvariablen.....33
Abbildung 17: Einstellung des Pfades.....34
Abbildung 18: Optionen der Remote Verbindung.....35
Abbildung 19: Hinzufügen eines Fahrzeugs.....36
Abbildung 20: Graphen Einstellungen.....42
Abbildung 21: Armaturen Einstellungen.....42
Abbildung 22: Änderung der Sleeptime.....43

5.2 Mathematische Modellbildung:

Variablenbezeichnung:

v	Geschwindigkeit
g	Fallbeschleunigung
ρ	Dichte
m_{PKW}	Masse des Fahrzeugs (Leergewicht)
A	Querschnitt (Angriffsfläche)
c_w	Luftwiderstandsbeiwert
r	Reifenrollradius
f_r	Rollwiderstandsbeiwert
d_{schalt}	Schaltdauer
n_{leer}	Leerlaufdrehzahl (für Startgeschwindigkeit)
n_{schalt}	Schaltdrehzahl
gz	Anzahl der Gänge
g(gz)	Feld von Gängen belegt mit Übersetzungen
d	Differentialübersetzung

Während der Fahrzeugbeschleunigung wirken folgende Kräfte auf das Fahrzeug:

Antriebskraft	$F_A = M_{mot}(n_{mot})\ \ i_G(G)\ \ i_D\ \ 1/r_{dyn}$
Luftwiderstand	$F_L = c_\omega\ \ \rho\ \ v^2\ \ A/2$
Rollwiderstand	$F_R = f_R\ \ m_{PKW}\ \ g$

Gesamtkraft F_{Ges}:

[1] $F_{Ges} = F_A - F_L - F_R$

[2] $F_{Ges} = m\ \ a$ mit $a = d^2x/d^2t$

aus [1] u. [2] folgt:

$F_A - F_L - F_R = m_{PKW}\ \ d^2x / dt^2$ Differentialgleichung 2. Ordnung!

$(M_{mot}(n_{mot})\ \ i_G(G)\ \ i_D\ \ 1/r_{dyn}) - (c_\omega\ \ \rho\ \ v^2\ \ A/2) - (f_R\ \ m_{PKW}\ \ g)$

$= m_{PKW}\ \ d^2x/dt^2$

Zu Lösung dieser Gleichung mit dem Eulerschen Integrationsverfahren, muss die Differentialgleichung 2. Ordnung in zwei Differentialgleichungen erster Ordnung umgewandelt werden:
Geschwindigkeit als zusätzliche Variable:

$V = dx/dt$

è $d^2x/dt^2 = dv/dt$

è $F_{Ges} = m \quad dv/dt$

è $dv/dt = F_{Ges} \quad 1/m_{PKW}$ [3]

è $dx/dt = v$ [4]

Mit [3] u. [4] kann jetzt mit dem Eulerschen Integrationsverfahren die Gleichung berechnet werden.

1. Schritt: k=0, t=0

Funktionsauswertung:

$x'_0 = v(t_0)$
$v'_0 = F_{Ges} \quad 1/m_{PKW}$

Integration:

$X_1 = x_0 + ?T \quad x'_0$
$v_1 = v_0 + ?T \quad v'_0$
$t = t + ?T$
$k = k + 1$

2. Schritt: k=1, t=?T

Funktionsauswertung:

$x'_1 = v_1$
$v'_1 = F_{Ges} \quad 1/m_{PKW}$

Integration:

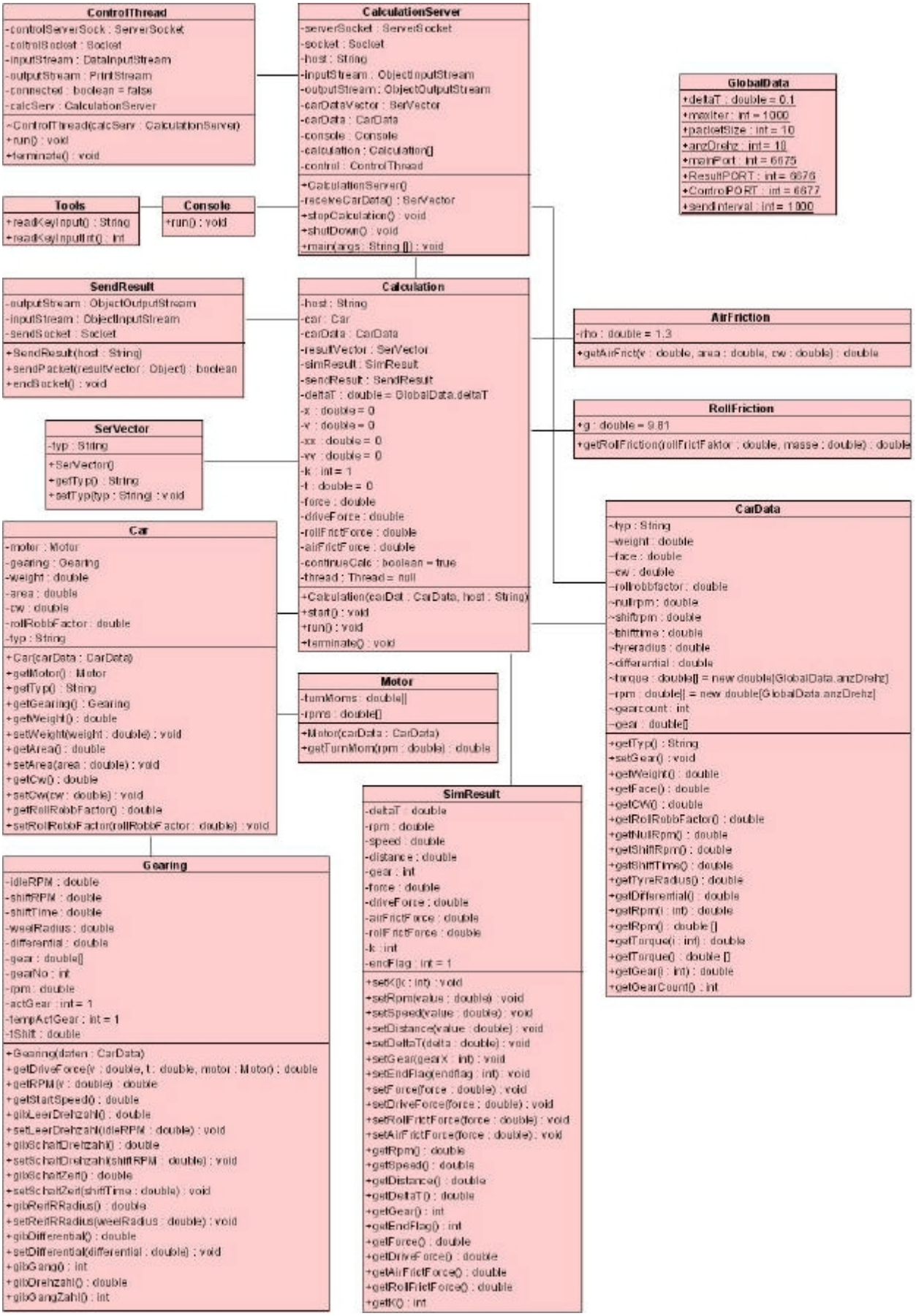
$X_2 = x_1 + ?T \quad x'_1$
$V_2 = v_1 + ?T \quad v'_1$

usw.

Die Integration wird solange fortgesetzt bis eine der Abbruchbedingungen erfüllt ist:

- Ablauf einer bestimmten Zeit t_D (solange $t<t_D$)
- Erreichen einer bestimmten Anzahl k_d von Berechnungen ($k< k_d$)
- Erreichen der Geschwindigkeit v_d ($v< v_d$)

5.3 Klassendiagramm Berechungsserver



5.4 Klassendiagramm Simulationsprogramm